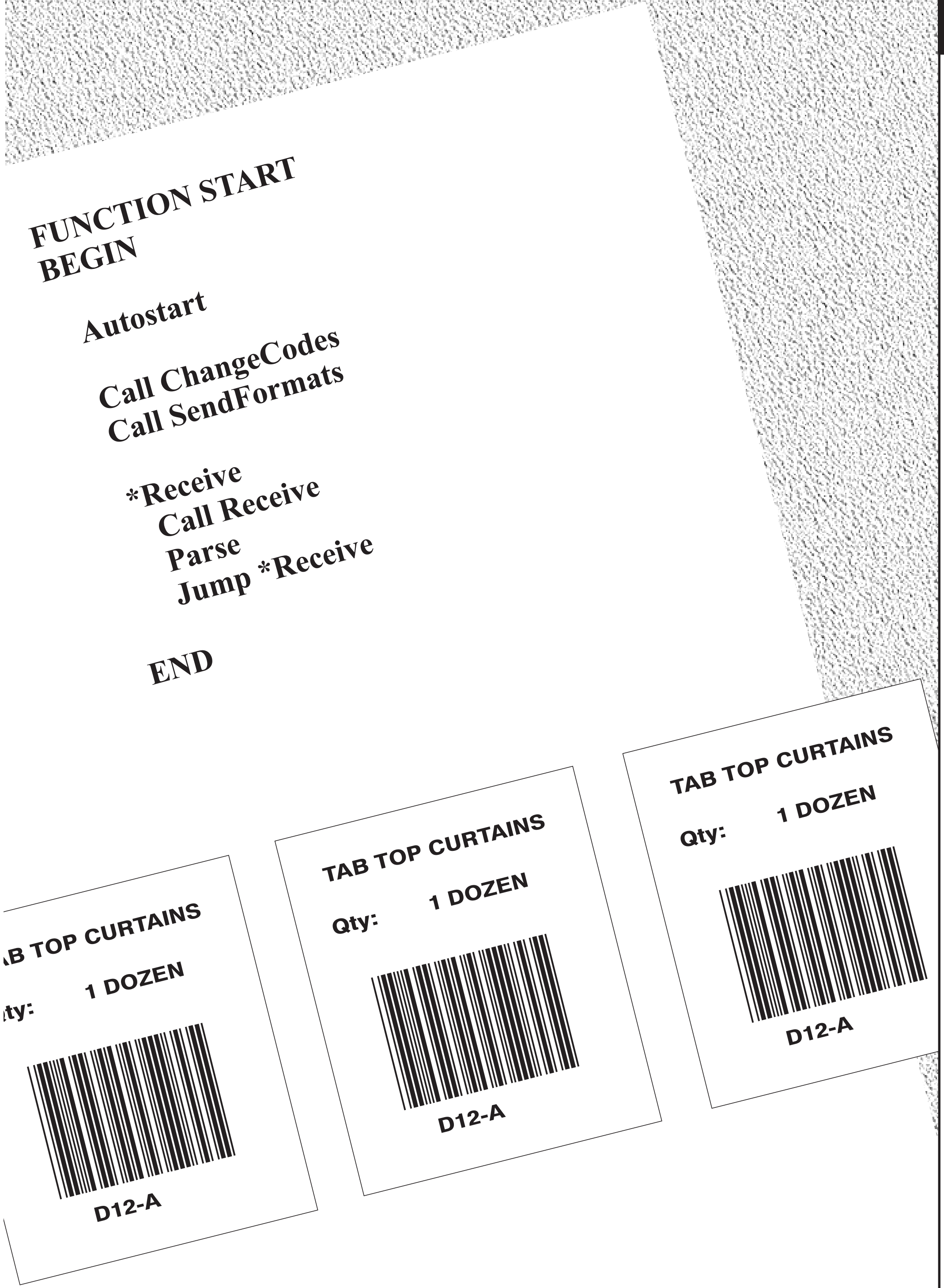
Programmer's Manual

Monarch®
9460™ ADK
Printer



AVERY DENNISON

TC9460ADKPM Rev. AC 1/08

Avery Dennison Printer Systems Division
170 Monarch Lane
Miamisburg, OH 45342

TABLE OF CONTENTS

OVERVIEW

1

The Application Development Kit II (ADK2) is a product for Microsoft® Windows® 95/98/Me/NT®/2000. It allows you to create an application program to run on the printer. You write the script with the ADK2 command language.

You can program the printer to:

- ♦ print labels or tags
- ♦ print data streams written for other printers

You can define lookup tables for the script running on the printer. It also allows you to define records such as temporary storage buffers.

This manual is written for the Monarch® Sierra Sport™ 2 9460™ printer. Refer to the printer's *Operator's Handbook* or *Quick Reference* for printer-specific information. Refer to the *Packet Reference Manual* for data stream information.

How to Use this Manual

This manual contains the following information.

Chapter 1 *Overview*	Introduces ADK2.
Chapter 2 *Using the Software*	Tells you how to use the software for entering, editing, compiling, and printing your script.
Chapter 3 *Printer Procedures*	Explains tasks done on the printer separate from the application.
Chapter 4 *Program Structure*	Tells you how to write the script's source code.
Chapter 5 *Command Reference*	Describes the commands you use to write your script.
Appendix A *Sample Script*	Lists a sample script.

A Review of Terms

Throughout this manual, you will see references to the different terms that you must be aware of before programming an ADK-version 9460 printer.

A *file* is a collection of related data, stored together in one unit. There are three types of files: scripts, formats, and lookup tables.

A *script* is a type of file. It is the source code for a program that runs on the printer.

A *project* is a collection of related files. The files can be a scripts, formats, or lookup tables. A project must have at least one script, but formats and lookup tables are optional.

An *application* is a project that has been built into a form executable by the printer.

USING THE SOFTWARE 2

This chapter explains how to

- ♦ start a new project.
- ♦ build a project into an application.
- ♦ download an application.

System Requirements

Here are the recommended system requirements.

	Recommended
Computer	Personal computer with Microsoft Windows 95/98/Me/NT/2000
Processor	Pentium – 150 Mhz
Memory	32 Meg
Disk space	5-10 Meg
Communications Port	Serial
Printer	Monarch Sierra Sport 2

Installing the Software

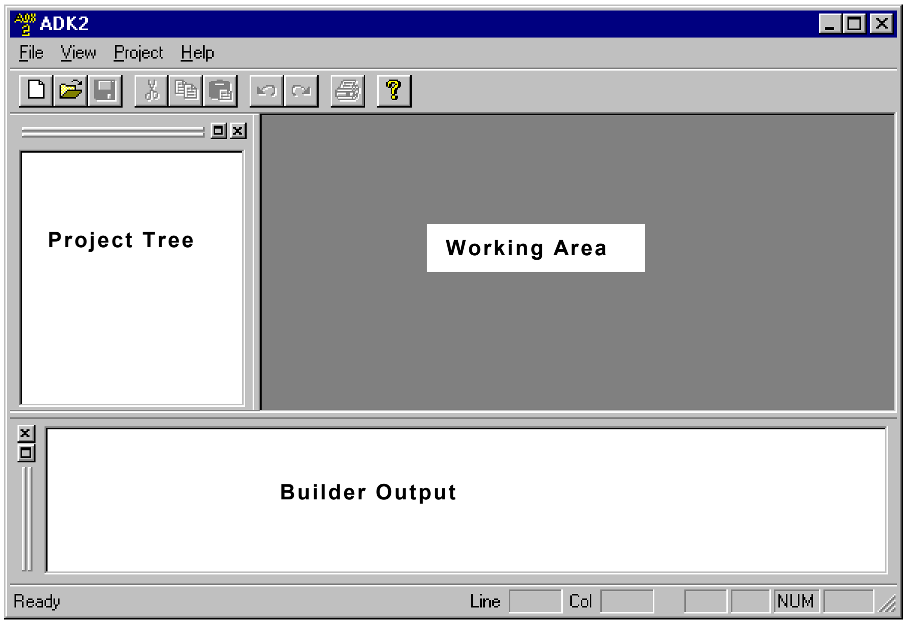
1. From the Start menu, run the file **SETUP.EXE**.
2. Respond to the prompts as necessary.

Connecting the Printer

Connect your printer to the PC using either a DB9 to 9 pin (part 12029314) or DB9 to 25-pin (part 12029315) serial communications cable. For more information about connecting the cable, refer to your *Operator's Handbook* or *Quick Reference*.

Getting Started

1. Start the ADK2 software. You will see

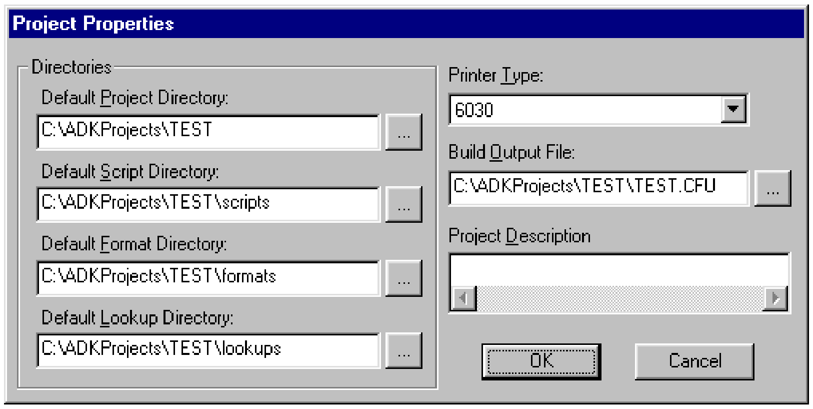


The screen has three major sections: the Project Tree, Working Area, and Builder Output. You can use the View menu to change which areas appear on your screen.

- ♦ The Project Tree lists all the files in the open project. See "About Projects" for more information.
- ♦ The Working Area is the text editor for the files in the project.
- ♦ The Builder Output lists any errors or messages that appear when you build the project.

2. Start a new Project: Select New from the Project Menu.

3. Enter a name for the Project. Press OK. You will see the Project Properties screen.

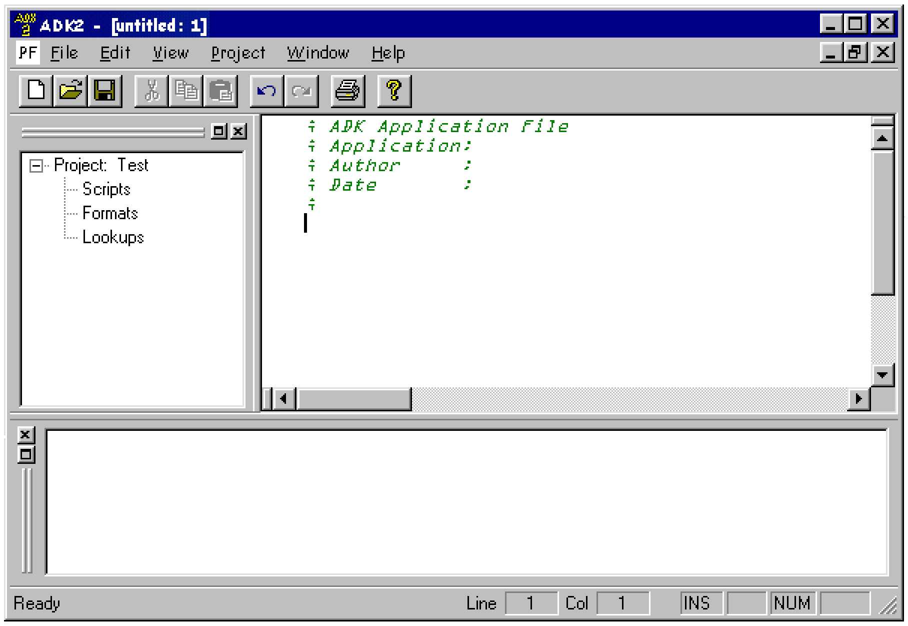


Default Project Directory: C:\ADKProjects\TEST
Default Script Directory: C:\ADKProjects\TEST\scripts
Default Format Directory: C:\ADKProjects\TEST\formats
Default Lookup Directory: C:\ADKProjects\TEST\lookups
Printer Type: 6030
Build Output File: C:\ADKProjects\TEST\TEST.CFU
Project Description

4. Accept the default directories or change the directories.

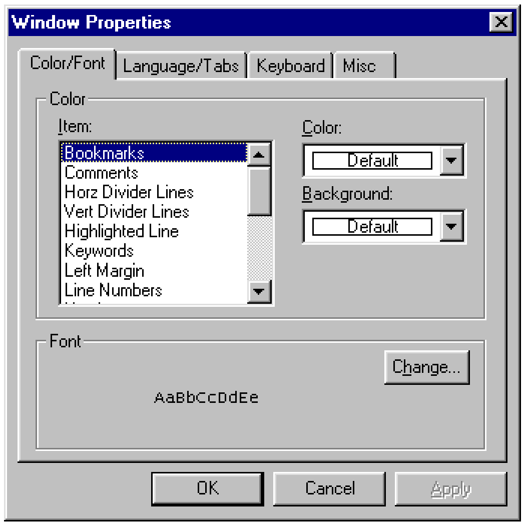
5. Select 9460 from the Printer Type box. Press OK. The project file structure is set up. You return to the Main screen.

6. Select New from the File Menu to start writing a script. Initial comments are automatically added in the Working Area of the screen.



```
; ADK Application File
; Application:
; Author      :
; Date        :
;
```

As you type your script, the ADK2 keywords appear in blue and the script text also appears in different colors, depending on what the text item is. To change the keyword colors, the text to upper or lower case, or show white space, right mouse click in the Working Area of the screen and make the appropriate selection. If you select Properties, you will see



7. Make any changes you want to the text color and tab sizes, enable line numbering, etc. Click OK when finished.

8. Finish writing your script.

Saving a File

Select Save from the File menu. The default sub-directory is \Scripts in the selected project directory. The file is saved with .CFS extension (configuration source). The first time you save the file, it will prompt you if you want to add this file as a script in the currently open project.

About Projects

The Project Tree lists all the files in the open project. The project tree contains the following directories: scripts, formats, and lookups.

Scripts Multiple scripts can be included in the \Scripts directory for use in the current project.

1. Highlight the \Scripts directory.
2. Right mouse click and select Add Files to Folder.
3. Locate and select the script to add. Click Open.

NOTE: One script must be marked as the Main script before building. Highlight the script. Right mouse click and select Mark as Main. You must use the INCLUDE command in the script to include the other scripts.

Example:

```
Define SCRATCH, 5000, A
INCLUDE c:\ADKProjects\MyStore\Scripts\price.cfs
Function Start
Begin
.
.
.
```

Formats Add format files to the \Formats directory for use in the current project.

1. Highlight the \Formats directory.
2. Right mouse click and select Add Files to Folder.
3. Locate and select the format to add. Click Open.

NOTE: You must use the LINKFILE command in the script to include the format.

Example:

```
Define SCRATCH, 5000, A
LINKFILE c:\ADKProjects\MyStore\Formats\shipping.fmt
Function Start
Begin
.
.
.
```

Lookups Add lookup tables to the \Lookups directory for use in the current project.

1. Highlight the \Lookups directory.
2. Right mouse click and select Add Files to Folder.
3. Locate and select the lookup table to add. Click Open.

   NOTE: If you do not use the LOOKUPDEF command in the script, when the script is downloaded, you are prompted for the lookup file.

```
Example:
Define SCRATCH, 5000, A
LOOKUPDEF c:\ADKProjects\MyStore\Lookups\prices.txt
Function Start
Begin
.
.
.
```

Building Projects

When a script has been marked as the Main script, you are ready to build.

1. Select Build from the Project Menu.
2. The Builder Output portion of the screen shows different types of messages: Build Successful, Build Aborted, Syntax Error, etc. A successfully built project file is saved with a .CFU extension.

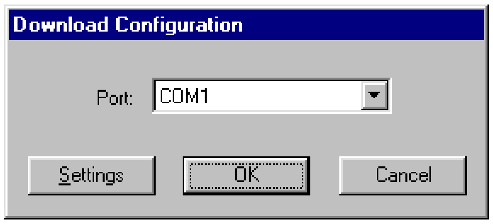
   NOTE: You can select Properties from the Project Menu to change file extension.
3. Specify the download settings.
4. Download the built file to the printer.

Changing the Download Settings

Before downloading a project to the printer, make sure the download settings at the PC match those at the printer.

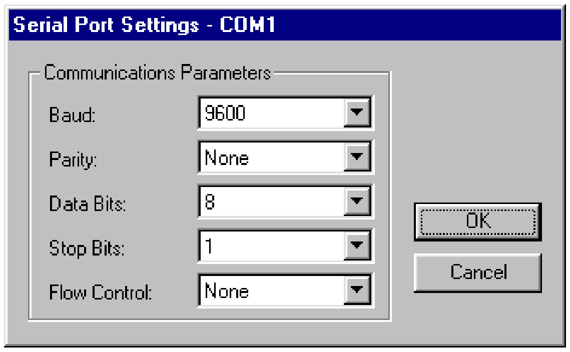
To change the PC's download settings:

1. Select Download Settings from the Project Menu.



2. Select the communications port (COM1, LPT1-2, or TCP/IP).
3. Click Settings.

If you select COM1 - COM4:

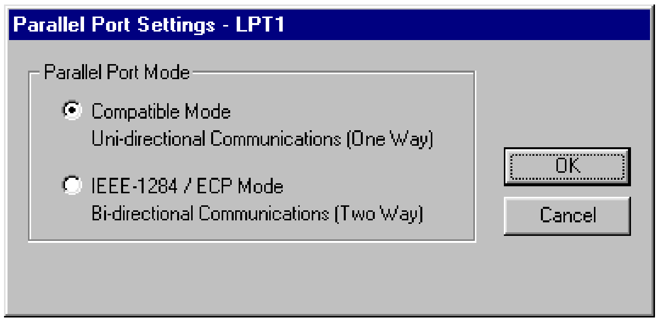


Baud: 9600
Parity: None
Data Bits: 8
Stop Bits: 1
Flow Control: None

4. Make changes as needed to the Baud, Parity, Data Bits, Stop Bits, and Flow Control. Click OK twice.

NOTE: Changing these parameters only affects your PC, not the connected printer. Refer to your printer's documentation for more information about changing the printer's communications parameters.

If you select LPT1 – LPT2:



(●) Compatible Mode
Uni-directional Communications (One Way)
( ) IEEE-1284 / ECP Mode
Bi-directional Communications (Two Way)

Compatible mode is for uni-directional communications. With this mode, you can send files to your printer, but you will not receive printer status information. Select this mode if you are unsure of your printer's parallel port configuration or your PC's parallel port configuration.

IEEE1284 mode is for bi-directional communications. With this mode, you can send files to your printer and receive printer status information, such as error messages.

Only select this mode if:

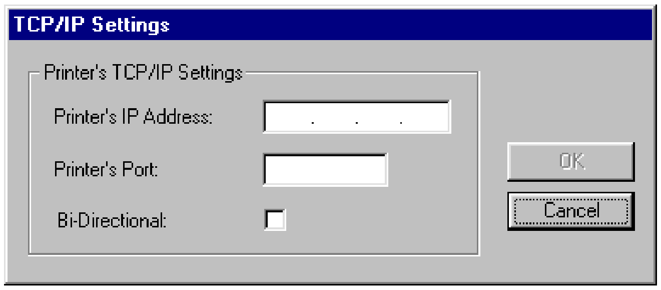
- your printer supports IEEE-1284 and it is enabled.
- your computer supports ECP mode and ECP mode is enabled on your computer's parallel port. This is typically selected in your computer's BIOS setup, which is normally accessed whenever you turn on your computer.

This screen appears differently for Microsoft Windows NT® and Windows® 2000 users.

Windows NT:	Use the Direct Memory Access (DMA) channel assigned to your LPT port. The DMA normally defaults to 3. This can be changed in your computer's BIOS setup.
For Windows 2000:	Use the Direct Memory Access (DMA) channel assigned to your LPT port. The DMA normally defaults to 3. Enable the LPT port's Interrupts using Device Manager.

5. Make a choice and click OK when finished.

If you select TCPIP:



4. Enter your printer's TCP/IP Address. See your System Administrator for more information.
5. Enter your printer's TCP/IP Port (typically 9100). See your System Administrator for more information.
6. Determine appropriate bi-directional setting:
    - ♦ Disabled/Unchecked is for uni-directional communications. With this mode, you can send files to your printer, but you will not receive printer status information. Disable/Uncheck this selection if you are unsure of your printer's parallel port configuration.
    - ♦ Enabled/Checked is for bi-directional communications. With this mode, you can send files to your printer and receive printer status information, such as error messages. Only select this mode if your printer is set for IEEE1284 mode. Refer to your printer's manual for more information.
7. Click OK to exit the Download Configuration screen.

Downloading a Project

After the project has been built, you are ready to download it to the printer.

1. Verify that the download settings are the same at both the PC and printer. See "Changing the Download Settings," for more information.
2. Select Download from the Project Menu. Messages appear as the file is downloaded to the printer.

Editing Existing Projects

1. Select Open from the Project Menu and locate the project file. It as a .CFP extension.
2. Make any changes to your script, format, or lookup file(s).
3. Save your changes.
4. Re-build the project.
5. Download the project to the printer.

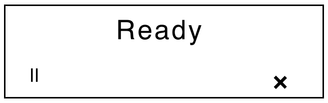
When you close the ADK2 software, it saves the current views and which project files are open. When you re-open the project, the software restores the views and the previously opened project files.

PRINTER PROCEDURES 3

Applications should be written so that they run continuously when the machine is on. However, there may be instances where you need to "go behind the scenes" to troubleshoot the printer, reload an application, perform maintenance or set parameters.

Displaying the Ready Prompt

Before doing anything, you must display the Ready prompt on the printer.



```
Ready
II              ×
```

How you do this depends on whether the printer has an application loaded.

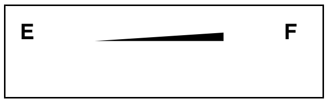
No Application

When there is no application in the printer, the Ready prompt appears automatically when you turn on the printer.

Loaded Application

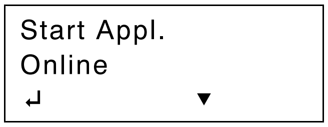
To display the Ready prompt with a loaded application:

1. Turn on the printer. In a moment, the battery charge indicator appears.



```
E               F
```

2. Press the key with the batter charge indicator on the screen. The following menu appears:



```
Start Appl.
Online
↵          ▼
```

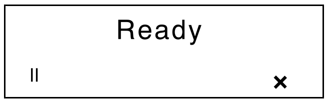
3. Choose **Online**. The Ready prompt appears.

NOTE: *Online* is different from Online Diagnostics in the tool box.

Accessing the Toolbox

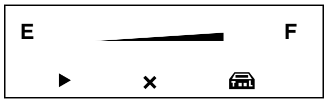
You may need to run diagnostic tests, perform maintenance or set parameters on the printers. To do this, access the toolbox, as follows:

1. From the Ready prompt, press the left ▲ button (under the II icon).



```
Ready
II              ×
```

2. The battery charge indicator appears.



```
E               F
   ▶     ×
```

3. Press the right ▲ button (under the 🖶 icon). The following menu appears.

```
Tool Box
Language
Exit
↲            ▼
```

4. Choose **Tool Box**.

Note: When you exit the tool box, the Ready prompt appears.

5. Use the tool box to perform the tasks you need to do. See the *System Administrator's Guide* for more information.

Loading Applications

After you use the tool box, you must restart the existing application or load a new one.

Restarting Existing Applications

To restart the application (after using the tool box):

1. From the Ready prompt, press the right ▲ key (under the × icon). The following menu appears.

```
Start Appl.
Online
↲            ▼
```

2. Choose Start Appl.

Loading New Applications

To load a new application, see Chapter 2, "Using the Software."

PROGRAM STRUCTURE

4

This chapter discusses program flow control, buffer definitions, and other useful information for writing your script.

Below is a sample of what a script may look like.

```
DEFINE TEMPORARY, RegPrice, 6, A
DEFINE TEMPORARY, NewPrice, 6, A

DEFINE PRINTER, PrtRegPrice, 7, A
DEFINE PRINTER, PrtNewPrice, 7, A

FUNCTION Start
BEGIN
   CALL InitApp
   CALL GetRegPrice
   CALL GetNewPrice
   CALL PrintTags
END

FUNCTION InitApp
BEGIN
   CLEAR Printer
   CLEAR RegPrice
   .
   .
   .
END
.
.
.
```

Functions

A function is an independent group of statements usually performing a specific task. You execute a function with the CALL command. See Appendix A, "Sample Script," for a sample script.

Rule: Each function must have a BEGIN and an END.

```
FUNCTION function-name
   BEGIN
     .
     function-body
     .
   END
```

Starting a Script

Every script has the primary function START. The START function is the starting point of your program. Script execution control starts with the first command in START, and stops when the last command in START is performed.

```
FUNCTION START
   BEGIN
      .
      program-body
      .
   END
```

Files and Buffers

The Lookup table is a collection of records. Data is stored in the printer as an ASCII flat file.

You can tell the printer how to store defined buffers in memory. You can define the following buffers:

- ♦ Scratch buffer
- ♦ Lookup table buffer
- ♦ Temporary storage buffer
- ♦ Printer buffer
- ♦ Array buffer

A buffer may contain up to 255 separate fields, each field being 1 to 999 bytes long.

Use the DEFINE command to specify the field definitions. Field-type, field-name, field-length, and data-type are the fields used to define the buffer. See Chapter 5, "Command Reference," for more information.

Lookup Table Definition

The lookup buffer is the working area for data downloaded to the printer. The lookup table definition tells the printer how the lookup records are received from the PC. The printer allocates buffer space for the record when it receives the definition.

The number of records stored depends on the size of each record and the script's size.

Temporary Storage Definition

The temporary storage buffer is used as a temporary storage for arithmetic operations and temporary variables.

Printer Definition

The printer buffer is used to store data to print.

Rule: The field lengths in the printer buffer must equal the length of the largest corresponding field in the formats. For example, if...

the length of Field 1 of Format 1 is 7
the length of Field 1 of Format 2 is 22
the length of Field 1 of Format 3 is 12

Then, the first field's length in the printer buffer must be 22.

Arrays

You can use an array to store data similar to temporary storage. An array is a series of elements with the same data type. Arrays can be either numeric or alphanumeric. You can access an element of an array by providing the array name and an index value. This index value can be a numeric literal, a numeric buffer-field, or the input buffer. For example, Prices [4] points to the Prices array's fourth element.

In addition to the information for the DEFINE command listed above, you must also list the number of elements in the array (the maximum index value). See Chapter 5, "Command Reference," for more information.

Scope of Field Names

Keep in mind the following information.

- You can access all variables globally.
- Field names and labels can be up to 255 characters long. However, the first 12 characters must be unique.

Script Flow Control

You can branch the flow of command control in different ways. The order in which the commands appear in the script controls the program's flow. At times, control is passed to another command through the use of valid labels, invalid labels, and the JUMP command. See Chapter 5, "Command Reference," for information about JUMP.

When a command fails, control passes to an invalid label, if you defined one. For example, the invalid label may show a message on the printer display. If the script does not identify an invalid label, control passes to the next line following the executing command. Similarly, when execution is successful, control passes to a valid label, if you defined one. And, if you did not define a valid label, control passes to the next line.

Rule: Precede all valid and invalid labels by an asterisk, (*). For example,

```
ADD CONTROL , TEMP1 , *ERROR2, *SUCCESS2
```

Comments in a Script

You must precede comments by a semicolon. The software treats them as a single white space and ignores them.

```
;**************************************************************
;*
;* Description
;* This is the main entry point of
;* the script.  Gets the Date and
;* then starts processing.
;*
;**************************************************************
```

Data Storage

Although you can define a buffer field as being numeric or alphanumeric, the printer stores both kinds of data as ASCII characters, as follows:

Data Type	Description
Alphanumeric	Sequences of **any** ASCII characters.
Numeric	Sequences of **numeric** ASCII characters. For example, the printer stores 91 as the two-byte alphanumeric string "91."

Data Coding

To streamline the amount of data you store or pass to and from the printer, you can encode the data.

For example, you could encode a number as high as 255 by storing the corresponding character from the ASCII chart. For example, 91 (a two-byte character string, according to printer data storage rules) could appear as [, the ninety-first character on the ASCII chart.

There are two commands you can use in your script when encoding and decoding data according to this method.

Command	Description
ASC	Takes an ASCII character and returns the number corresponding to it on the ASCII chart.
CHR	Takes a number from 0-255 and returns the corresponding character on the ASCII chart.

Consider the following code sample.

```
DEFINE TEMPORARY, QTY1, 3, A     ; Alpha Temp. field
DEFINE TEMPORARY, QTY2, 3, N     ; Numeric Temp. field
MOVE "}", QTY1                   ; Now contains "}"
ASC QTY1, QTY2                   ; Decodes "}" to 125
INC QTY2                         ; Increments 125 to 126
CHR QTY2, QTY1                   ; Encodes 126 to "~"
```

This sample demonstrates how to decode a number, use the number in a computation, and encode the result back to a character.

COMMAND REFERENCE 5

This chapter lists, in alphabetical order, the commands you use to write your script. Each command is discussed in detail to include the correct syntax.

Programming Conventions

The commands use the following conventions.

KEYWORDS	You must type the upper-case text. **CALL** *function-name*
Place holders	Text in italics are place holders. **CLEAR** *item*
[optional]	Optional items appear in brackets. **CHECK** *item* [,[*<MI>invalid label*] [,*valid label*]]
Example	Text in bold courier font are examples of the command in use. **ADD WHOLESALE , TEMP2**
*label	Text with an asterisk, "*", is a label signifying a place to jump to in the script. **ADD CONTROL , TEMP1 , *ERROR2**
Repeating Items	Horizontal ellipsis dots following an item in a syntax description indicate more of the same item may appear. **FETCH COMM**
Missing Items	Vertical ellipsis dots used in examples and syntax descriptions indicate a portion of the code is omitted. Ex. **ADD WHOLESALE, TEMP2** . . . **ADD TEMP2, TEMP1**

Field Names

The logical field names used in the command sections are examples. For example, TEMP1 is used throughout this chapter as an example of a temporary buffer field name.

Keywords

The following keywords are reserved by the compiler. Do not use them as identifiers.

1200	DTRDTE	NONE
1200	19.2K	2400
4800	9600	ADD
APPEND	APPVERSION	ARGREAD
ARRAY	ASC	AUTOSTART
AVAILABLEDATA	BACKLIGHT	BATTERY
BAUDRATE	BEEP	BEGIN
BITCLEAR	BITMASK	BITSET
BITSHIFT	BITTEST	BSEARCH
CALL	CASE	CHARTYPE
CHECK	CHR	CLEAR
CLOSECOMM	COMM	COMM2
COMPARE	CONCAT	CONTINUOUS
CSTRIP	CURRENT	CURRENCY
DATABITS	DATACOLLECT	DATACOLLECTFILE
DATATYPE	DATE	DATELEN
DEC	DEFINE	DELAY
DELIMITER	DISABLE	DISPLAY
DIVIDE	DOWNLOAD	DTRDTE
ECHOBELL	ELSE	ELSEIF
ENABLE	END	ENDIF
ENDSWITCH	ENDWHILE	ENTER
EVEN	EXECUTE	EXIT
F1	F2	F3
F4	F5	F6
FAILSAFE	FETCH	FIELDLEN
FIXDATA	FORMAT	FUNCTION
GENERATE	GET	HEADER
HOTKEY	IF	IMAGEBUFFER
IMAGEFIELD	INC	INCLUDE
INPUT	INPUTTEMPLATE	INSERT
JUMP	KEYBOARD	LABELCOUNT

LEFT	LINKFILE	LINKFMT
LOCATE	LOOKUP	LOOKUPDEF
LOOKUPFILE	LOOKUPSIZE	LOWER
LSTRIP	MACRO	MARK
MID	MOVE	MULTIPLE
MULTIPLY	NONE	NUMBERPRINTED
ODD	ONDEMAND	OPENCOMM
PACKRECORDS	PAD	PARITY
PARSE	PRINT	PRINTER
PROMPTS	QUERY	RAM
RAVAIL	RCLOSE	READ
RECORDDELETE	RESPONSE	RESTORESCREEN
RETURN	REVVID	RIGHT
ROPEN	RREAD	RSTRIP
RTSCTS	RWRITE	SAVESCREEN
SCANLEN	SCANNER	SCRATCH
SEEK	SETDATE	SHUTDOWN
SKIP	SPACE	START
STATUSPOLLING	STOPBITS	STRIPS
SUB	SUSPEND	SWITCH
SYSSET	SYMBOL	TEMPORARY
TOKEN	TRIGGER	TRIGGERENABLE
TSTRIP	UPLOAD	UPLOADDEF
UPPER	VALIDATE	WHILE
WRITE	XONXOFF	

NOTE: Not all of these keywords apply to the 9460 printer; however, they are still reserved by the compiler.

Special Characters

The following special characters are reserved for the printer. Do not use them in your script.

{	left brace
_	underscore
|	pipe or split vertical bar
}	right brace
~	tilde
\	backslash
`	grave accent

However, you can use these characters in a string with quotation marks.

Use the tilde character (~) along with the corresponding ASCII code in strings to represent non-printable characters. For example, ~013 represents a carriage return.

The tilde sequence also works for using a double quote in a quoted string in a command parameter. For example, to move a double quote (") to the scratch buffer, enter:

```
MOVE "~034", SCRATCH
```

Script Flow

Script flow branches out to other functions and labels, depending on whether a command was successful or if it failed.

When a label is defined...

- If a command was successful and a valid label is defined, control passes to that label.
- If a command fails and an invalid label is defined, control passes to that label.

When a label is NOT defined...

- If a command was successful, control passes to the next line.
- If a command fails, control passes to the next line.

Functional Relationships

Some commands logically work together or are related in function. The commands are discussed in the following functional groups.

Math Commands

ADD	Adds the numeric values of two fields.
DEC	Decrements numeric fields.
DIVIDE	Divides the contents of one field by the contents of another.
INC	Increments numeric fields.
MULTIPLY	Multiplies the contents of one field by the contents of another field.
SUB	Subtracts the contents of one field from the contents of another field.

Script Control Commands

CALL	Calls a section of code as a subroutine.
CHECK	Checks the status of system parameters.
DELAY	Delays the current script for a specified time.
EXIT	Leaves the current script. If the printer re-enters the script, control passes to the script's first line.
IF	Performs a series of one or more commands based on the existence of a condition.
JUMP	Transfers control to a label.
RETURN	Exits a subroutine.
SWITCH	Branches to a set of commands, based on the value of a variable.
SYSSET	Sets system parameters.
WHILE	Repeats a series of one or more commands based on the existence of a condition.

Compiler Directives

DEFINE	Defines the field definitions for the buffers.
INCLUDE	Inserts the source statements in the file into the current script.
LINKFILE	Links formats to the script so they can download to the printer.
MACRO	Defines or invokes a program for a repeating process.

Data Manipulation Commands

ARGREAD	Extracts an argument from a comma-delimited string.
ASC	Converts ASCII data from a numeric format to an alphanumeric format.
BITCLEAR	Sets the specified bit to zero.
BITMASK	Allows bit logical operations on buffers.
BITSET	Sets the specified bit to one.
BITSHIFT	Allows bits within a value to be arithmetically shifted left or right.
BITTEST	Checks the specified bit to see if the bit is a one or a zero.
CHARTYPE	Allows you to limit the character type for an input buffer.
CHR	Converts ASCII data from an alphanumeric format to a numeric format.
CLEAR	Clears buffers or files.
COMPARE	Compares the contents of two fields.
CONCAT	Appends the contents of one field to another.
CSTRIP	Extracts specific characters for a string.
DATATYPE	Restricts the type of data for the GET command.
FIELDLEN	Places the length of one field into another.
GENERATE	Creates a check digit.
INSERT	Inserts data from one buffer into another.
LEFT	Extracts the left-most characters from a string.
LOWER	Converts characters in a field to lower-case.
LSTRIP	Strips specified left-most characters from one field and copies the remaining characters to another.
MID	Extracts a sub-field from a string.
MOVE	Copies contents of one field to another field.
PAD	Adds characters to a field to fill it out.
PARSE	Processes an MPCL data stream.
RIGHT	Extracts the right-most characters from a string.
RSTRIP	Strips specified right-most characters from one field and copies the remaining characters to another.
TOKEN	Extracts character-delimited sub-fields from a string.
TSTRIP	Strips characters from a field based upon a template.
UPPER	Converts characters in a field to upper-case.
VALIDATE	Validates a check digit.

File Management Commands

APPVERSION	Sets the script name and version number.
BSEARCH	Performs a binary search on a sorted lookup table for a record containing a specific value.
QUERY	Searches a lookup file to find a record containing a specific value.
READ	Copies the current record from the lookup file into the appropriate working buffer.
SEEK	Positions the current record within the lookup file.

Input/Output Commands

AUTOSTART	Executes the script immediately after download is complete.
AVAILABLEDATA	Checks the communications port for available data.
CLOSECOMM	Closes the communications port.
DISABLE	Turns off a particular hot key.
ENABLE	Turns on a particular hot key.
FETCH	Retrieves one character from the communications port and places it in the input buffer.
FIXDATA	Defines fixed data for an input buffer.
GET	Retrieves data from the communications port.
HOTKEY	Defines a particular hot key.
LABELCOUNT	Tracks the number of labels printed.
LOCATE	Moves the cursor to a particular position on the printer’s Screen.
OPENCOMM	Opens the communications port.
PRINT	Prints the printer buffer's contents in the format specified.
RESTORESCREEN	Re-displays the saved contents of the screen.
SAVESCREEN	Saves the screen’s current contents.

ADD

Purpose — Adds the numeric values of two fields.

Syntax — ADD *buffer-field1* , *buffer-field2* [, [*invalid label*] [, *valid label*]]

Process — The ADD command sums *buffer-field1* and *buffer-field2* and places the result into *buffer-field2*.

The *buffer-field* fields can be one of the following:

Buffer-field 1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only:	Number prefixed by the number sign (#)

Rules: Both fields must be numeric.
The DEFINE command defines the index.

Optional Fields — *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example 1 — This example adds the contents of WHOLESALE to TEMP2. Control passes to the next line.

```
ADD WHOLESALE,TEMP2
```

Example 2 — This example assigns TEMP1 the sum of CONTROL and TEMP1. If TEMP1 overflows, control passes to the invalid label *ERROR2. If TEMP1 does not overflow, control passes to the next line.

```
ADD CONTROL,TEMP1,*ERROR2
```

See Also — DEC
INC
SUB
MULTIPLY
DIVIDE

APPVERSION

Purpose	Sets the version string of the ADK application.
Syntax	APPVERSION "*string1*", "*string2*"
Process	The APPVERSION command has *string1* appear on the screen's first line and *string2* on the screen's second line.
	String1 and *string2* can be up to 16 characters long.
Example	This example displays **AP11** on the screen's first line and **VER 1.0** on the second line.

```
APPVERSION "AP11", "VER 1.0"
```

ARGREAD

Purpose	Extracts field data from one field and places it in another.
Syntax	ARGREAD *raw-data*, *destination*, *index* [, [*invalid label*] [, *valid label*]]
Process	The ARGREAD command extracts data from *raw-data* and places it in *destination*.

The *raw-data, destination, and index* fields can be one the following:

Raw-Data, Destination, and Index	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Raw-data field only: String	ASCII string delimited by double quotes.
Index field only: Number	Number prefixed by a number sign (#). Range is 1-99.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example Assuming PARAMLIST contains **SSN,Name,Item**, the following example extracts SSN and places it in the PARAM1 variable.

```
ARGREAD PARAMLIST, PARAM1, #1
```

ASC

Purpose	Converts numeric data to alphanumeric data.
Syntax	ASC *int-field*, *asc-field*

Process: The ASC command converts numeric data from *int-field* and places the resulting alphanumeric data in *asc-field*.

The *int-field and asc-field* fields can be one the following:

Int-field and Asc-field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name [index]	Array Buffer Field
SCRATCH	Scratch Buffer
Int-field only: Number	Number prefixed by a number sign (#).

Example The following example converts numeric data from the TAINT field, converts it into alphanumeric data, and stores the result in TAASCII.

```
ASC TAINT, TAASCII
```

See Also: CHR

AUTOSTART

Purpose	Starts the application immediately after it is downloaded to the printer.
Syntax	AUTOSTART
Process	The AUTOSTART command starts the application immediately after it is downloaded to the printer. Place it anywhere in the application code, but use it only once.
Example	This example specifies that the application should start immediately after download to the printer.

```
AUTOSTART
```

AVAILABLEDATA

Purpose	Checks for data at a device.
Syntax	AVAILABLEDATA *device* [, [*invalid label*] [, *valid label*]]
Process	The AVAILABLE command checks for data at a *device*.

The *device* field can be one the following:

Device	**Description**
KEYBOARD	Keypad
COMM	Communications Port

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example In the following example, control passes to code at label *CHARLN if the application detects data on the communications port.

```
AVAILABLEDATA COMM, , *CHARLN
```

BITCLEAR

Purpose	Sets the specified bit to zero.
Syntax	BITCLEAR *buffer-field, bit-position* [, [*invalid label*] [, *valid label*]]
Process	The BITCLEAR command sets the specified bit by *bit-position* in *buffer-field* to zero. *Bit-position* can be 0 to 15. If *bit-position* is out of range and *invalid label* is defined, control passes to that label.

Buffer-field and *bit-position* can be one of the following:

Asc-field* and *Int-field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only:	
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes
SCRATCH	Scratch buffer

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example This example sets bit number two of TEMP1 to zero.

```
BITCLEAR TEMP1, #2
```

See Also
BITMASK
BITSET
BITSHIFT
BITTEST

BITMASK

Purpose
Allows bit logical operations on buffers.

Syntax
BITSET *operation*, *buffer-field1*, *buffer-field2* [, [*invalid label*] [, *valid label*]]
use the above syntax for logical AND/OR or logical exclusive OR.

BITMASK *operation*, *buffer-field1* [, [*invalid label*] [, *valid label*]]
use the above syntax for Invert.

Process
The BITMASK command allows bit logical operations on *buffer-field1*. AND/OR and exclusive OR take the value in *buffer-field2* and logically combine it with the contents of *buffer-field1*. The result is stored in *buffer-field1*.

The INVERT operation inverts all bits in *buffer-field1*. If *buffer-field1* or *buffer-field2* and *invalid label* are defined, control passes to that label. If the operation is successful and *valid label* is defined, control passes to that label.

RULE: Both *buffer-field1* and *buffer-field2* must be numeric.

Operation can be one of the following:

Operation	**Description**
AND	Logical And
OR	Logical Or
XOR	Logical Exclusive Or
INVERT	Invert all bits

Buffer-field1 and *buffer-field2* can be one of the following:

Buffer-field1* and *Buffer-field2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only:	
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes
SCRATCH	Scratch buffer

Optional Fields
Invalid and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example
This example strips the high 8 bits from TEMP1.

```
BITMASK AND, TEMP1, #255
```

See Also
BITCLEAR
BITSET
BITSHIFT
BITTEST

BITSET

Purpose	Sets the specified bit to one.
Syntax	BITSET *buffer-field1*, *bit-position* [, [*invalid label*] [, *valid label*]]
Process	The BITSET command sets the specified bit by *bit-position* in *buffer-field* to one. *Bit-position* can be 0 to 15. If *bit-position* is out of range and *invalid label* is defined, control passes to that label.

RULE: Both *buffer-field1* and *bit-position* must be numeric.

Buffer-field and *bit-position* can be one of the following:

Buffer-field* and *bit-position	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only:	
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes
SCRATCH	Scratch buffer

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example This example sets bit number two of TEMP1 to one.

```
BITSET TEMP1, #2
```

See Also
BITCLEAR
BITMASK
BITSHIFT
BITTEST

BITSHIFT

Purpose Allows bits within a value to be arithmetically shifted left or right.

Syntax BITSHIFT *direction*, *buffer-field1*, *count* [, [*invalid label*] [, *valid label*]]

Process The BITSHIFT command allows bits within a value to be arithmetically shifted left or right. Shifts *count* bits in *buffer-field* in the direction specified by *direction*. *Count* can be 1 to 16. If *buffer-field* contains an invalid value or the count field is out of range and invalid label is defined, control passes to that label. If the operation is successful and *valid label* is defined, control passes to that label.

RULE: Both *buffer-field1* and *count* must be numeric.

Direction can be one of the following:

Direction	**Description**
LEFT	Shifts bits left
RIGHT	Shifts bits right

Buffer-field1 and *count* can be one of the following:

Buffer-field1* and *count	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only:	
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes
SCRATCH	Scratch buffer

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example This example shifts the bits in TEMP1 once to the left, which has the effect of doubling the value.

```
BITSHIFT LEFT, TEMP1, #1
```

See Also BITCLEAR
BITMASK
BITSET
BITTEST

BITTEST

Purpose — Checks the specified bit to see if the bit is a one or a zero.

Syntax — BITTEST *buffer-field1*, *bit-position* [, [*invalid label*] [, *valid label*]]

Process — The BITTEST command checks the specified bit by *bit-position* in *buffer-field1*. *Bit-position* can be 0 to 15. If the bit specified by *bit-position* is zero (cleared) and *invalid label* is defined, control passes to that label. If the bit specified by *bit-position* is one (set) and *valid label* is defined, control passes to that label.

Buffer-field1 and *bit-position* can be one of the following:

Buffer-field1* and *bit-position	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only:	
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes
SCRATCH	Scratch buffer

Optional Fields — *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example — This example checks bit number two of TEMP1 and if it is zero (cleared), control passes to CLEARED. If the bit is one (set), control passes to the next line.

```
BITTEST TEMP1, #2, *CLEARED
```

See Also — BITCLEAR
BITMASK
BITSET
BITSHIFT

BSEARCH

Purpose	Performs a binary search on a sorted lookup table to find a record containing a specific value.
Syntax	BSEARCH *lookup-field*, *value* [, [*invalid label*] [, *valid label*]]
Process	The BSEARCH command searches *lookup-field* for *value*. The script determines which lookup table to use by the field you specify (every field name must be unique over all lookup tables).
Optional Fields	If the search is successful, the pointer points to the record and control passes to *valid label* (if defined). If the search is unsuccessful, the pointer is undefined and control passes to *invalid label* (if defined). Otherwise, control passes to the next line.

NOTE: You must sort the lookup table before downloading it to the printer.

Lookup-field is the search field's logical name in the lookup table.

Value is the value you are searching the field for and can be one of the following:

Value	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	A number prefixed by the number sign (#)
String	A one-character ASCII string delimited by double quotes

Lookup-field and *value* must have the same data type.

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning this chapter.

Example This example searches CONTROL_ID for the input buffer's contents. If no match is found, control transfers to the *ERROR_ID label.

```
BSEARCH CONTROL_ID, INPUT, *ERROR_ID
```

CALL

Purpose Calls a section of code as a subroutine.

Syntax CALL *function-name* [(*param1, param2, ...,paramX*)]

Process The CALL command executes an out-of-line function. After execution, control returns to the command following the CALL command. The CALL function allows parameters to passed to the called function. The called function *function-name* must have a DEFINE **LOCAL** for local storage for each parameter that is passed to it. The parameters will be placed from left to right with the leftmost parameter placed in the first DEFINE **LOCAL** variable.

Rules: You may nest up to 25 CALL commands.

The *param* fields can be one of the following:

Param	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
String	ASCII string delimited by double quotes.
Number	Number prefixed by a number sign (#).
Scratch	Scratch buffer

Example 1 This example calls the subroutine COMPUTE_TAX.

```
CALL COMPUTE_TAX
```

Example 2 This example shows the use of local variables. The function ADDNUM has two local variables defined that receive the parameters passed from the call. The first parameter (#5) is placed in the TfirstNum field and the second parameter (#6) is placed in the TsecondNum field. The TSum variable is defined as a global Temporary variable and is accessible from any functions in the script.

```
DEFINE TEMPORARY, TSum, 10, N
FUNCTION START
BEGIN
.
.
.
CALL ADDNUM (#5, #6)
.
.
.
END

FUNCTION ADDNUM
BEGIN
DEFINE LOCAL,TfirstNum, 10, N
DEFINE LOCAL, TsecondNum, 10, N
ADD TfirstNum, TsecondNum
MOVE TsecondNum, Tsum
END
```

See Also RETURN

CHARTYPE

Purpose	Allows you to limit the character type for an input buffer.
Syntax	CHARTYPE *type* , *buffer-field1*
Process	The CHARTYPE command restricts the character type applied to an input buffer using *type* and *buffer-field1*.

NOTE: This command does not affect the FETCH command.

Type describes the contents of *buffer-field1*. It can be one of the following:

Type	**Description**
S	Set of allowable characters for input field
T	Template mask for input buffer

Buffer-field1 contains either a set of allowable characters for the input field or a template mask. It can be one of the following:

Buffer-field1	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	A number prefixed by the number sign (#)
String	ASCII string delimited by double quotes

Valid characters for a template mask are:

Type	**Description**
*	Any valid character
#	Numeric
@	Alpha character
-	Skip input for fixed data

Example This example restricts a template input field to accept one alphanumeric character, four numeric characters, fixed data followed by two more numeric characters. For example, $9999.99.

```
CHARTYPE T,"@####_##"
```

See Also FIXDATA

CHECK

Purpose: Checks the status of a specified system parameter.

Syntax: CHECK *item* [, *buffer-field*] [, [*invalid label*] [, *valid label*]]

Process: The CHECK command checks the status of *item*.

Item can be one of the following:

Item	**Description**
BATTERY	Battery Voltage
COMM	Communications Port
PRINT	Print

Buffer-field is required if *item* is COMM or PRINT. It is a field where the status of the parameter is returned. Following are the possible status values:

Item	***Value***	***Description***
COMM	0	OK
	1	User Aborted
	400	Invalid Packet Received
	406	Response Time-out
	410	Parity Error
	411	Communications Error (framing, overrun)
	413	Input Buffer Full (XON not acknowledged)
PRINT	0	Good
	1	User Aborted
	750	Hot Printed
	751	Jam
	762	Battery Voltage Too Low to Print
	770	Motor not Ready
	771	Format not Found

Buffer-field can be one of the following:

Buffer-field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example This example checks the battery level. If the level is low, control of the application branches to the *LOWBATTERY label.

```
CHECK BATTERY, *LOWBATTERY
```

CHR

Purpose — Converts alphanumeric data to numeric data.

Syntax — CHR *asc-field*, *int-field*

Process — The CHR command converts *asc-field* (containing alphanumeric data) to a numeric format, placing the result in *int-field*.

Asc-field and Int-field contain the data to translate and the translated data, respectively. They can be one of the following:

Asc-field and Int-field	**Description**
INPUT	Input Buffer
SCRATCH	Scratch Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Asc-field only: String	ASCII string delimited by double quotes

Example — This example takes alphanumeric data from the TAASCII field, converts it into numeric data, and stores the result in the TAINT field.

```
CHR TAASCII, TAINT
```

See Also — ASC

CLEAR

Purpose Deletes data from data items.

Syntax CLEAR *item*

Process The CLEAR command deletes data from *item*. Control always passes to the next line.

NOTE: You must define a buffer before you can clear it.

Item is the data item to clear. It can be one of the following:

Item	**Description**
PRINTER	Printer Buffer
INPUT	Input Buffer
DISPLAY	Printer's Screen
TEMPORARY	Temporary Buffer Record
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
NUMBERPRINTED	Number of labels printed
INPUTTEMPLATE	Input Template, Chartype, and DataType settings
SCRATCH	Scratch Buffer
COMM	Communications port

Example 1 This example clears the Printer Buffer and passes control to the next line.

```
CLEAR PRINTER
```

Example 2 This example clears temporary buffer field TEMP1 and passes control to the next line.

```
CLEAR TEMP1
```

CLOSECOMM

Purpose	Closes either the primary or secondary communications port.
Syntax	CLOSECOMM *commport*
Process	The CLOSECOMM command closes communications port referenced by *commport*. It can contain 1 for the primary port or 2 for the secondary port.

Commport is the communications port to close. It can be one of the following:

Commport	**Description**
INPUT	Input Buffer
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Number	A number prefixed by a pound (#) sign.

Example This example closes the primary communications port.

```
CLOSECOMM #1
```

See Also OPENCOMM

COMPARE

Purpose Compares the contents of two fields.

Syntax COMPARE *buffer-field1* , *modifier* , *buffer-field2* [, [*invalid label*] [*, valid label*]]

Process The COMPARE command compares the two buffer fields, based on *modifier*.

Modifier can be one of the following:

Modifier	**Description**
GT	Greater than operator
GE	Greater than or equal to operator
LT	Less than operator
LE	Less than or equal to operator
EQ	Equal to operator

The *buffer-field* fields can be one of the following:

Buffer-field1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
String	ASCII string delimited by double quotes
Number	A number prefixed by the number sign (#)

Rule: B*uffer-field1* and *buffer-field2* must be the same type. For example, if *buffer-field1* is numeric, *buffer-field2* must also be numeric.

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

Rules: If the comparison is true, control passes to *valid label* or to the next line if there is no *valid label*.
If the comparison is false, control passes to *invalid label* or to the next line if there is no *invalid label*.

Example This example compares TEMP1 and TRUCK_ID for equality. If they are equal, control passes to *TRUCK_IN. If they are not equal, control passes to *JUMP_5.

```
COMPARE TEMP1,EQ,TRUCK_ID,*JUMP_5,*TRUCK_IN
```

CONCAT

Purpose	Appends the contents of one field to another.
Syntax	CONCAT *source* , *destination* [, [*invalid label*] [, *valid label*]]
Process	The CONCAT command copies *source*'s contents to the end of *destination*'s contents. *Source*'s contents do not change.

Source is the data to append. *Destination* is the resulting data. These variables can be one of the following:

Source* and *Destination	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Source only:	
String	ASCII string delimited by double quotes
Number	A number prefixed by the number sign (#)
Destination only:	
SCRATCH	Scratch Buffer Field

You can concatenate numeric fields and alphanumeric fields in any combination.

Optional Fields: Invalid and valid labels are discussed in “Script Flow" at the beginning of this chapter.

Example This example appends the SKU to the end of BC_FIELD.

```
CONCAT SKU,BC_FIELD
```

CSTRIP

Purpose	Strips data from a field.
Syntax	CSTRIP *field-buffer1* , *field-buffer2* [, [*invalid label*] [, *valid label*]]

Process: The CSTRIP command strips data specified in *field-buffer2* from *field-buffer1*. These variables can be one of the following:

Field-buffer1 and Field-buffer2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Field-buffer2 only: String	ASCII string delimited by double quotes.

Optional Fields: Invalid and valid labels are discussed in “Script Flow" at the beginning of this chapter.

Example This example removes all dashes from the SHIP_NO field.

```
CSTRIP SHIP_NO, "-"
```

See Also: RSTRIP
LSTRIP
TSTRIP

DATATYPE

Purpose Restricts the type of data the GET statement can retrieve.

Syntax DATATYPE *data-type*

Process The DATATYPE command restricts the GET statement to only read data of type *data-type*. *Data-type* can contain one of the following values.

Data-type	**Description**
NUMERIC	Numeric Only (0-9)
ALPHA	Alpha only (A-Z, a-z)
SYMBOLS	Symbols only
ALPHANUMERIC	Alphanumeric (0-9, A-Z, a-z)
NUMSYM	Numeric and Symbols
ALPHASYM	Alpha and Symbols
ALPHANUMSYM	Alphanumeric and Symbols
ALL	All characters accepted (00-FFh)

Example This example removes all dashes from the SHIP_NO field.

```
DATATYPE ALPHANUMERIC
```

DEC

Purpose Decrements numeric fields by one.

Syntax DEC *buffer-field* [, [*invalid label*] [*, valid label*]]

Process The DEC command decrements *buffer-field*. A translation error occurs if the script decrements an alphanumeric field.

Buffer-field is one of the following:

Buffer Field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field

Rule: You can decrement only numeric fields.

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

If you decrement an uninitialized field, control passes to *invalid label*.

Example This example decrements TEMP_SKU and passes control to the next line.

```
DEC TEMP_SKU
```

See Also ADD
INC
SUB
DEC
MULTIPLY
DIVIDE

DEFINE

Purpose Defines the field definitions for the buffers.

Syntax To define a SCRATCH buffer...

```
DEFINE  field-type , field-length , data-type
```

To define TEMPORARY or PRINTER buffers...

```
DEFINE  field-type , field-name , field-length [ , data-type ]
```

To define an ARRAY buffer...

```
DEFINE  field-type , field-name , field-length , number-of-elements [ , data-type ]
```

To define a LOOKUP buffer...

```
DEFINE  field-type , [ logical-name ] field-name , field-length [ , data-type ]
```

Process The DEFINE command defines temporary, lookup, printer, array, and scratch buffer fields.

Field-type can be one of the following:

Buffer Field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
SCRATCH	Scratch Buffer

Field-name is the field's logical name and is under the same restrictions as any other identifier.

Field-length is the buffer field's size in bytes. Enter a value from 1 to 2800. If you're defining a scratch buffer, the maximum is 65535.

NOTE: Although individual lookup table fields can be up to 2800 bytes long, lookup table records cannot exceed 128K.

If a DEFINE TEMPORARY statement is placed inside the BEGIN-END pair of a function, that variable can only be referenced within that function and not by any other function.

Optional Fields *Logical-name* is used to define multiple lookup tables. Each name must be unique (over all lookup tables used by the script) and in parentheses. The default name is lookup.

Number-of-elements is required when the *field-type* is set to ARRAY.

Data-type is the kind of data the buffer field holds. Enter A (for alphanumeric) or N (for numeric). The default is A.

NOTE: If *field-type* is PRINTER, *data-typ*e must be A.

Example 1 This example defines the temporary buffer field CURR_QTY as a numeric field with a length of 4 bytes.

```
DEFINE TEMPORARY,CURR_QTY,4,N
```

Example 2 This example shows the use of local variables. The function ADDNUM has two local variables defined that receive the parameters passed from the call. The fist parameter (#5) is placed in the TfirstNum field and the second parameter (#6) is placed in the TsecondNum field. The Tsum variable is defined as a global Temporary variable and is accessible from any functions in the script.

```
DEFINE TEMPORARY, Tsum, 10, N
FUNCTION START
BEGIN
.
.
.
CALL ADDNUM (#5, #6)
.
.
.
END



FUNCTION ADDNUM
BEGIN
DEFINE LOCAL, TfirstNum, 10, N
DEFINE LOCAL, TsecondNum, 10, N
ADD TfirstNum, TsecoundNum
MOVE TsecondNum, Tsum
END
```

DELAY

Purpose Delays the current script for a specified time.

Syntax DELAY #*interval*

Process The DELAY command suspends the printer's current script for the number of tenths of seconds specified by *interval*. The *interval* range is 1 - 255.

Interval can be one of the following:

Interval	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	Number prefixed by a number sign (#)

Rule: The *interval* must be numeric.

Example 1 This example suspends the current script for two seconds.

```
DELAY #20
```

Example 2 This example suspends the current script for the number of tenths of seconds in TIMEOUT.

```
DELAY TIMEOUT
```

DISABLE

Purpose	Turns off the specified hot keys.
Syntax	DISABLE *hotkey1*[, *hotkey2*][, *hotkey3*]
Process	The DISABLE command turns off the specified hot keys. You must turn on the hot keys (with the ENABLE command) before using this command.

Hotkey1, hotkey2, and hotkey3 can be one of the following:

Hotkey1, Hotkey2, and Hotkey3	**Description**
F1	Function Key 1
F2	Function Key 2
F3	Function Key 3
ALL	All function keys

Example This example disables the F1, F2, and F3 hot keys.

```
DISABLE F1, F2, F3
```

See Also ENABLE
HOTKEY

DIVIDE

Purpose: Divides the contents of one field by the contents of another.

Syntax: DIVIDE *buffer-field1* , *buffer-field2* [, [*invalid label*] [*, valid label*]]

Process: The DIVIDE command divides *buffer-field1* by *buffer-field2* and inserts the quotient into *buffer-field2*. This command performs integer division and truncates the remainder.

Buffer-field1 contains the dividend while *buffer-field2* is the divisor. These variables can be one of the following:

Buffer Field1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	Number prefixed by a number sign (#)

Rules: You cannot use two numeric literal fields. For example,

- ♦ If *buffer-field1* contains a numeric literal, *buffer-field2* must contain a field.
- ♦ If *buffer-field2* contains a numeric literal, *buffer-field1* must contain a field.

When you use a numeric literal, the script places the result in the field that is not a numeric literal.

Optional Fields: Invalid and valid labels are discussed in “Script Flow" at the beginning of this chapter.

Example 1 This example divides the contents of WHOLESALE by the contents of TEMP2. The quotient is inserted into TEMP2. Control passes to the next line.

```
DIVIDE WHOLESALE,TEMP2
```

Example 2 This example divides the contents of CONTROL by the contents of TEMP1, inserting the quotient into TEMP1. If an overflow condition occurs, control passes to *ERROR2.

```
DIVIDE CONTROL,TEMP1,*ERROR2
```

Example 3 This example divides the contents of PRICE by 100. This operation is a method of converting cents to dollars. If an overflow condition occurs, control passes to *ERROR2.

```
DIVIDE PRICE,#100,*ERROR2
```

See Also: MULTIPLY

ENABLE

Purpose Turns on the specified hot keys.

Syntax DISABLE *hotkey1*[, *hotkey2*][, *hotkey3*]

Process The DISABLE command turns on the specified hot keys. You must turn on the hot keys (with the ENABLE command) before using this command (the default is off).

Hotkey1, hotkey2, and hotkey3 can be one of the following:

Hotkey1, Hotkey2, and Hotkey3	**Description**
F1	Function Key 1
F2	Function Key 2
F3	Function Key 3
ALL	All function keys

Example This example disables the F1, F2, and F3 hot keys.

```
ENABLE F1, F2, F3
```

See Also DISABLE
HOTKEY

EXIT

Purpose	Leaves the current script.
Syntax	EXIT
Process	The EXIT command returns control back to normal printer operation unless you specify AUTOSTART.
	To restart the script, enable the script through the printer's control panel.
Example	This example shows the script's termination.

```
EXIT
```

FETCH

Purpose — Retrieves one character from up to two sources and places it in the Input Buffer.

Syntax — FETCH *src1* [, *src2*] [, [*invalid label*] [*, valid label*]]

Process — The FETCH command retrieves one character from *src1* and optionally, *src2*. It places these characters in the Input Buffer.

NOTE: The DATATYPE and CHARTYPE commands do not affect this command.

Src1 and *src2* can be one of the following:

Src1 **and** ***Src2***	**Description**
COMM	Communications port
KEYBOARD	Keypad

Characters retrieved from the keyboard will be either **1**, **2**, or **3**, depending on which hot key was pressed.

Optional Fields — Invalid and valid labels are discussed in “Script Flow" at the beginning of this chapter.

Example — This example retrieves one character from the communications port and passes control to the next line.

```
FETCH COMM
```

See Also — GET

FIELDLEN

Purpose Places the length of a field into another field.

Syntax FIELDLEN *buffer-field1* , *buffer-field2* [, [*invalid label*] [*, valid label*]]

Process The FIELDLEN command calculates the length of *buffer-field1* and places it in *buffer-field2*.

The *buffer-field* fields can be one of the following:

Buffer Field1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only:	
String	ASCII string delimited by double quotes.
Number	Number prefixed by a number sign (#)

Rule: *Buffer-field2* must be numeric.

Optional Fields *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

Example 1 This example places the length of WHOLESALE into TEMP2. Control passes to the next line.

```
FIELDLEN WHOLESALE,TEMP2
```

FIXDATA

Purpose — Defines fixed data for an input buffer.

Syntax — FIXDATA *buffer-field1*

Process — The FIXDATA command defines fixed data for the input buffer. Use this command with the CHARTYPE command, which provides a template. *Buffer-field1* contains a string inserted into the input buffer.

Buffer-field1 can be one of the following:

Buffer Field1	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
String	ASCII string delimited by double quotes
Number	Number prefixed by a number sign (#)

Rules: Spaces in a string represent fixed spaces.
An underscore, "_", is a place holder for variable data.

Example — This example creates a template for a telephone number. The CHARTYPE command could define the variable characters as numeric.

```
FIXDATA "(___) ___-___"
```

See Also — CHARTYPE
TSTRIP

GENERATE

Purpose — Generates a check digit.

Syntax — GENERATE *buffer-field*, *type* [, [*invalid label*] [*, valid label*]]

Process — The GENERATE command generates a check digit for the value in *buffer-field*. *Type* specifies the check digit scheme to use.

Buffer-field and *type* can be one of the following:

Buffer-field and Type	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Buffer-field only: Logical Field Name (Field1) [Index]	Array Buffer Field
Type Only: Number	Number (from 1-24) prefixed by a number sign (#).

Rule: When it is a buffer field, *type* must be numeric.

Following are the meanings of each value *type* can have.

1	Reserved	**13**	Custom Check Digit 9
2	Sum of Digits	**14**	Custom Check Digit 10
3	Sum of Products	**15**	UPCA Check Digit
4	Reserved	**16**	UPCE Check Digit
5	Custom Check Digit 1	**17**	EAN8 Check Digit
6	Custom Check Digit 2	**18**	EAN13 Check Digit
7	Custom Check Digit 3	**19**	LAC Check Digit
8	Custom Check Digit 4	**20**	Code 39 Check Digit
9	Custom Check Digit 5	**21**	MSI Check Digit
10	Custom Check Digit 6	**22**	Postnet Check Digit
11	Custom Check Digit 7	**23**	UPC Price Check Digit
12	Custom Check Digit 8	**24**	EAN Price Check Digit

Optional Fields — *Invalid* and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

Example — This example generates a check digit in the input buffer by using the Sum of Digits check digit scheme.

```
GENERATE INPUT, #2
```

GET

Purpose Retrieves data from up to two input devices.

Syntax GET *src1 [, src2]*, *minimum*, *maximum* [*, type*] [*,* [*invalid label*] [*, valid label*]]

Process The GET command retrieves data from *src1, and optionally, src2,* and places it in the input buffer.

Src1 and *src2* can be one of the following:

Src1 and Src2	**Description**
COMM	Communications port
KEYBOARD	Keypad

Minimum and *maximum* represent the field length. If *minimum* is 4 and *maximum* is 6, a valid entry for that field is 4 to 6 characters. The valid range for *minimum* and *maximum* is 0 - 512 characters. These fields can be one of the following:

Minimum* and *Maximum	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	Number prefixed by a number sign (#)

NOTE: Use the FETCH command if both *minimum* and *maximum* equal zero.

Optional Fields *Type* specifies the input's character type as:

Type	**Description**
N	Numeric only
A	Alphanumeric

Type overrides what you set up with the DATATYPE and CHARTYPE commands.

Alphanumeric is the default for *type* (only when you do not set up a *type* with DATATYPE or CHARTYPE).

Invalid and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

Examplef This example retrieves data from the communications port.

```
GET COMM,#0,#255
```

See Also FETCH

HOTKEY

Purpose	Defines hot keys.
Syntax	HOTKEY *key*, *function-name*

Process

The HOTKEY command defines *key*, specifying that the application should call *function-name* when the operator presses it.

Key can be one of the following:

Key	**Description**
F1	Function Key 1
F2	Function Key 2
F3	Function Key 3

Example

This example specifies that, when the operator presses F3, the application calls the QUERY_LOOKUP function.

```
HOTKEY F3, QUERY_LOOKUP
```

See Also

DISABLE
ENABLE

IF

Purpose

Performs a series of one or more commands if a certain condition exists.

Syntax

```
IF  buffer-field1 comparison buffer-field2
.
.
.
[ELSEIF buffer-field2 comparison buffer-field4]
.
.
.
[ELSE]
.
.
.
ENDIF
```

Process

The IF command directs script flow by determining if a condition or series of conditions exist. A condition is specified by comparing buffer fields. If the comparison is true (the condition exists), the script executes the commands on the lines following the condition. If the comparison is not true (the condition does not exist), control passes to the

- ♦ line after the ENDIF.
- ♦ next ELSEIF.
- ♦ first line after the ELSE.

You may nest IFs, but every IF must have a corresponding ENDIF.

NOTE: Do not use IF inside a macro.

The *buffer-fields* can be one of the following:

Buffer-fields	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
String	ASCII string delimited by double quotes
Number	Number prefixed by a number sign (#)

Comparison can be one of the following:

Comparison	**Description**
=	Equals
==	Equals
<>	Not equal
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

Optional Fields ELSEIF provides another set of commands to execute if another condition exists. For example,

```
IF NAME == "JOHNSON"
   INC JCOUNT
ELSEIF NAME == "SMITH"
   INC SCOUNT
ENDIF
```

counts the number of records where NAME is Johnson or Smith. The first condition is (NAME equals Johnson). The second condition is (NAME equals SMITH).

Use ELSE to provide a final set of commands to execute if no conditions exist.

Example This example checks the value of TASTATE. If it contains OHIO, the first MOVE command copies TASTATE to PASTATE. Otherwise, the second MOVE command copies the string "Out of State" to PASTATE.

```
IF TASTATE == "OHIO"
   MOVE TASTATE,PASTATE
ELSE
   MOVE "Out of State",PASTATE
ENDIF
```

See Also COMPARE
SWITCH
WHILE

INC

Purpose
Increments numeric fields by one.

Syntax
INC *buffer-field* [*,* [*invalid label*] [*, valid label*]]

Process
The INC command increments *buffer-field*.

Buffer-field can be one of the following:

Buffer Field1	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field

Rule: *Buffer-field* must be numeric. A translation error occurs if you increment an alphanumeric field.

Optional Fields
Invalid and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

If you increment an uninitialized field, the software sets *buffer-field* to 1 and control passes to *invalid label*. If the field overflows, control also passes to *invalid label*.

Example
This example increments COUNT01.

```
INC COUNT01
```

See Also
ADD
DEC
SUB
MULTIPLY
DIVIDE

INCLUDE

Purpose	Inserts another source file into the script.
Syntax	INCLUDE *pathname*
Process	The INCLUDE command signals the compiler to insert the source statements located in the file *pathname*, into the current script.
Rule:	Nested INCLUDE statements are not allowed. But, multiple INCLUDE statements in one file are allowed.

Example 1 This example inserts the source file TRUCKIN.ULT into the current script.

```
INCLUDE TRUCKIN.ULT
```

Example 2 This example inserts the source file SPECIAL.ULT into the current script.

```
INCLUDE C:\PROGS\SAMPLE\SPECIAL.ULT
```

INSERT

Purpose Inserts data from one buffer into another.

Syntax INSERT *overwrite-flag , buffer-field1 , buffer-field2 , position* [*,* [*invalid label*] [*, valid label*]]

Process The INSERT command inserts data from *buffer-field1* into *bufferfield2* at a specified *position*.

Overwrite-flag can be one of the following:

Overwrite-flag	**Description**
I	Insert data into field, pushing existing data over
O	Overwrite existing data in field

The *buffer-field* fields can be one of the following:

Buffer Field1* and *position	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 and *position* only: Number	Number prefixed by a number sign (#)
Buffer-field1 only: String	ASCII string delimited by double quotes

Rule: *Position* must be numeric.

Optional Fields If there is not enough room in *buffer-field2*, control passes to *invalid label*.

Invalid and *valid labels* are discussed in “Script Flow" at the beginning of this chapter.

Example This example inserts "This text will be inserted" into ASZPRICE at position POSNUM.

```
INSERT I,"This text will be inserted",ASZPRICE,POSNUM
```

See Also VALIDATE

JUMP

Purpose — JUMP transfers control to another location.

Syntax — JUMP *label*

Process — The JUMP command unconditionally transfers control to the specified label. If the script is re-entered, control passes to the script's first line.

Rule: You cannot jump out of a function.

Example — This example transfers control to the label *REQUEST_SKU.

```
JUMP *REQUEST_SKU
```

See Also — CALL

LABELCOUNT

Purpose	Sets a field to the current number of labels printed.
Syntax	LABELCOUNT *buffer-field1* [*,* [*invalid label*] [*, valid label*]]
Process	The LABELCOUNT command sets *buffer-field1* to the current number of labels printed.

Buffer-field1 field can be one of the following:

Buffer-field1	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example This example sets NUMOFLABELS to the number of labels the printer has printed.

```
LABELCOUNT NUMOFLABELS
```

LEFT

Purpose	Extracts the left-most character from a string.
Syntax	LEFT *buffer-field1* , *buffer-field2* , *length* [, [*invalid label*] [, *valid label*]]

Process

The LEFT command extracts the left-most characters from *buffer-field1* and copies them into *buffer-field2*. *Length* specifies the number of characters.

The *buffer-field* fields can be one of the following:

Buffer-field1 2* and *Length	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 and *length* only: Number	Number prefixed by a number sign (#)
Buffer-field1 only: String	ASCII string delimited by double quotes

Rule: *Length* must be numeric.

Optional Fields

Invalid and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example 1

This example extracts the five left-most characters from SHIP_NO and copies them to SKU.

```
LEFT SHIP_NO,SKU,#5
```

Example 2

This example extracts the NUMCHARS left-most characters from SHIP_NO and copies them to SKU.

```
LEFT SHIP_NO,SKU,NUMCHARS
```

See Also

LSTRIP
MID
RIGHT
RSTRIP

LINKFILE

Purpose	Links formats, files, or packets to the script.
Syntax	For files or packets created using a text editor. LINKFILE *file-name*
Process	The LINKFILE command links formats to the script. You can include any number of files in the download datastream. The LINKFILE command downloads formats, files, or packets created using a text editor. These commands add a line to the .CFU file's header which tells the transfer program to download the file specified by *format-name|file-name*. If you do not specify a path, the transfer program looks for a format in the \PLATFORM\FORMATS directory.
Rule:	Link files before FUNCTION START.

Example 1 This example downloads CHCKDGIT.PKT (created in MPCL with a text editor) to the printer.

```
LINKFILE CHCKDGIT.PKT
```

LOCATE

Purpose	Moves the cursor to a specified position on the printer's screen.
Syntax	LOCATE *row-position, col-position*
Process	The LOCATE command moves the cursor to the (*row-position*, *col-position*) position on the printer's screen. The range for *row-position* is 1-3. For *col-position* the ranges are 1-15 (for rows 1 and 2) and 1-20 (for row 3).

Row-position and col-position can be one of the following:

Source	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Number	A number prefixed by the number sign (#).

Example This example moves the cursor to the first row and second column of the screen.

```
LOCATE #1, #2
```

LOWER

Purpose	Converts characters in a field from upper-case to lower-case.
Syntax	LOWER *source*
Process	The LOWER command converts characters in *source* to lower-case characters.

Source can be one of the following:

Source	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field

Example This example converts any upper case characters in TEMP_SKU to lower-case characters.

```
LOWER TEMP_SKU
```

See Also UPPER

LSTRIP

Purpose — Strips characters from a field, and copies the remaining characters to another field.

Syntax — LSTRIP *field-buffer1, field-buffer2, field-buffer3* [*,* [*invalid label*] [*, valid label*]]

Process — The LSTRIP command strips the left-most characters from *field-buffer1* and copies the remaining characters to *field-buffer2*. *Field-buffer3* is the number of characters to strip.

Field-buffer1, *field-buffer2*, and *field-buffer3* can be one of the following:

Field-buffer1, Field-Buffer2, and Field-Buffer3	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Field-buffer1 and *Field-buffer2* only: String	ASCII string delimited by double quotes.
Field-buffer3 only: Number	Number prefixed by a number sign (#)

Optional Fields — *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example — This example strips the five left-most characters from the SHIP_NO field and copies the remaining characters to the SKU field.

```
LSTRIP SHIP_NO, SKU, #5
```

See Also — TSTRIP
RSTRIP
CSTRIP

MACRO

Purpose

Defines or invokes a single command the software expands to multiple commands during script translation.

Syntax

To define the macro ...

```
MACRO macro-name
BEGIN
macro-body
END
```

To invoke the macro ...

```
macro-name arg1 , arg2 , ... , arg99
```

Process

The MACRO command defines or invokes a macro. A macro is a single command the software expands to multiple commands during script translation. Each time a macro command appears, the software inserts the commands it generates into the script.

NOTE: **Do Not** use IF, SWITCH, or WHILE inside a macro.

Defining the Macro ...

The *macro-name* is an identifier naming the macro. The *macro-body* contains the commands defining what the macro does. The keywords BEGIN and END define *macro-body*'s boundary and limit the scope of control transfer to within the boundary.

Keep macros in a separate macro file you include in the source script using the INCLUDE command.

Rule: You must define macros before invoking them.

Invoking the Macro ...

The macro matches arguments. The first argument replaces %1, the second argument replaces %2, and so on, up to %99 arguments.

Labels are handled differently in macros. The label names inside the macro body should use this form:

```
*macro-label-name$
```

where *macro-label-name* is a unique name for the macro. The label can be up to eight characters. This restriction helps avoid duplicate labels if a macro appears within a function more than once.

As the compiler expands each macro ...

- it expands the labels.
- it expands each dollar sign ($) into a unique three-digit number.

Example This example defines a macro (PTRIDLE) to check the status of the printer.

```
DEFINE TEMPORARY, tEnqStatus, 3
DEFINE TEMPORARY, tPrinterOK, 1, N

MACRO PTRIDLE
BEGIN
MOVE    #0, %1
CHECK   ENQSTATUS, tEnqStatus
COMPARE tEnqStatus, EQ, "A@", *PI_END_$
MOVE    #1, %1
*PI_END_$
END
```

MID

Purpose	Extracts a sub-field from a string.
Syntax	MID *buffer-field1* , *buffer-field2* , *start* , *length* [*,* [*invalid label*] [*, valid label*]]
Process	The MID command extracts a sub-field from *buffer-field1* and copies it into *buffer-field2*; starting with the *start* position and extracting *length* number of characters.

Buffer-field1, *buffer-field2*, *start*, and *length* can be one the following:

Buffer-field1, 2, Start* and *Length	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only: String	ASCII string delimited by double quotes
Buffer-field1, start, and *length* only: Number	Number prefixed by a number sign (#)

Rules: *Length* and *Start* must be numeric.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exception is as follows:

If *buffer-field2* overflows, *start* is greater than *length*, or *invalid label* is defined, control passes to that label.

Example 1 This example extracts a five-character substring starting at position 5 of LOCATION and copies it into TEMP1.

```
MID LOCATION,TEMP1,#5,#5
```

Example 2 This example extracts a substring of LengthNum characters starting at position StartNum of LOCATION and copies it into TEMP1.

```
MID LOCATION,TEMP1,StartNum,LengthNum
```

See Also LEFT
LSTRIP
RIGHT
RSTRIP

MOVE

Purpose Copies data between fields.

Syntax MOVE *source*, *destination* [, [*invalid label*] [, *valid label*]]

Process The MOVE command copies data between fields. The contents of *source* replaces the contents of *destination* with no effect on *source*.

Source can be one of the following:

Source	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only: String	ASCII string delimited by double quotes
Buffer-field1, start, and *length* only: Number	Number prefixed by a number sign (#)
SCRATCH	Scratch Buffer

Destination can be one of the following:

Destination	**Description**
INPUT	Input Buffer
DISPLAY	The Printer's Screen
SCRATCH	Scratch Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1, start, and *length* only: Number	Number prefixed by a number sign (#)
Buffer-field1 only: String	ASCII string delimited by double quotes

You can move a numeric field into an alphanumeric field. However, you cannot move an alphanumeric field into a numeric field.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example 1 This example copies the data from the Input Buffer to CONTROL_ID.

```
MOVE INPUT,CONTROL_ID
```

MULTIPLY

Purpose Multiplies the contents of one field by the contents of another.

Syntax MULTIPLY *buffer-field1* , *buffer-field2* [, [*invalid label*]
[, *valid label*]]

Process The MULTIPLY command multiplies *buffer-field1* by *buffer-field2*, inserting the product into *buffer-field2*.

The maximum value for the *buffer-field1*, *buffer-field2*, and the result is 429,496,795.

The *buffer-field* fields can be one the following:

Buffer-field1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only: Number	Number prefixed by a number sign (#)

Rule: The *buffer-field* fields must be numeric.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example This example multiplies PRICE by TEMP1, inserting the product into TEMP1. If TEMP1 overflows, control passes to *ERROR2. If TEMP1 does not overflow, control passes to the next line.

```
MULTIPLY PRICE,TEMP1,*ERROR2
```

See Also DIVIDE
INC
SUB
ADD
DEC

OPENCOMM

Purpose	Opens either the primary or secondary communications port. The port stays open until it is closed with CLOSECOMM.
Syntax	OPENCOMM *commport*, *timeout*

Process

The OPENCOMM command opens the communications port referenced by *commport* (**1** for the primary port or **2** for the secondary port).

Timeout defines the length of time (0-255, in seconds) that the printer waits for data during a GET or FETCH. If a timeout occurs, control passes to the GET or FETCH invalid label. A *timeout* of 0 means the port waits indefinitely for data.

Commport and *timeout* must be numeric and can be one of the following:

Commport and Timeout	**Description**
INPUT	Input Buffer
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Number	A number prefixed by a pound (#) sign.

Example

This example opens the primary communications port, and times out after 120 seconds.

```
OPENCOMM #1, #120
```

See Also CLOSECOMM

PAD

Purpose Pads data in a field.

Syntax PAD *direction* , *pad-field* , *pad-character* , *max-length*

Process The PAD command pads data in *pad-field*, in the direction specified by *direction*, with *pad-character*. *Max-length* indicates the field's length. For example, if the data is seven characters and the length is ten, three characters are added to the field.

Direction can be one of the following:

Direction	**Description**
L	Pad left
R	Pad right

Pad-field, *pad-character*, and *max-length* can be one of the following:

Pad-field, Pad-character,* and *Max-length	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Pad-field only: Logical Field Name (Field1) [Index]	Array Buffer Field
Pad-character and *max-length* only: Number	Number prefixed by a number sign (#)
Pad-character only: String	ASCII string delimited by double quotes. Must be one character long

Rule: If *max-length* is a buffer field, it must be numeric.

Example 1 This example inserts asterisks (*) to the left of the data in PRICEFIELD.

```
PAD L,PRICEFIELD,"*",LPRICE
```

Example 2 This example inserts blanks to the right of the data.

```
PAD R,LDESC," ",#2
```

PARSE

Purpose	Processes an MPCL data stream in the scratch buffer.
Syntax	PARSE [[*invalid label*] [*, valid label*]]
Process	The PARSE command invokes the printer's MPCL parser to analyze and process the scratch buffer's contents.

In general, the PARSE command will out perform (speed to label out) the PRINT command. If you have the option of using either command (PARSE or PRINT), PARSE is the better option.

NOTE: You must place an MPCL data stream in the scratch buffer before calling this command.

Avoid using the PARSE command to send individual characters; use the CONCAT command to append data into the scratch buffer. Then send all the data at once using the PARSE command.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example This example moves an MPCL data stream to the scratch buffer, then processes the data stream.

```
MOVE "{F,1,A,N,E,200,200,″FMT1″¦″, SCRATCH
CONCAT "C,146,50,0,10,2,1,B,L,0,0,"PAT'S PARTS",1¦", SCRATCH
CONCAT "T,1,10,V,100,50,0,1013,3,1,B,L,0,0,1¦", SCRATCH
CONCAT "T,2,15,V,80,25,0,10,1,1,B,L,0,0,1¦", SCRATCH
CONCAT "L,V,67,1,0,180,10,"" ¦", SCRATCH
CONCAT "B,3,12,F,12,43,1,2,50,1,L,0¦}", SCRATCH
PARSE
```

NOTE: You cannot nest double quotes. You must use ~034 instead of a double quote. In the above example, use ~034FMT1~034 for "FMT1".

PRINT

Purpose — Prints the Printer Buffer's contents, by a source field, in the format specified.

Syntax — PRINT [CONTINUOUS] #*format-number,* [quantity]
[*,* [*invalid label*] [*, valid label*]]

Process — The PRINT command images and prints the format specified by *format-number*. *Format-number* contains a format number between 0 and 999. Numbers greater than 255 cannot be constants. If *format-number* equals 0, the same image prints. Use this method to avoid reimaging the data.

Quantity represents the number of labels to print. The printer pauses before printing each label. However, the printer does not pause when you use CONTINUOUS and *quantity* is greater than 1; it prints one strip with the number of labels in *quantity*. *Quantity* can be 1-99 (the default is 1).

Format-number and *quantit*y can be one of the following:

Format-number* and *quantity	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Number	Number prefixed by a number sign (#)

Rule: *Format-number* and *quantity* must be numeric.

Optional Fields — *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example — This example prints the Printer Buffer's contents using Format 2 and then passes control to the next line if successful. If the operator presses an exception key, control passes to *Exception.

```
PRINT #2,*Exception
```

QUERY

Purpose Searches the lookup file to find a specified record.

Syntax QUERY *buffer-field1* , *comparison* , *buffer-field2* [*,* [*invalid label*] [*, valid label*]]

Process The QUERY command searches the lookup file to find a record containing a specific value.

NOTE: If multiple records contain the value, the command reads the first record fitting the criteria.

Buffer-field1 specifies the buffer to search and can be one of the following:

Buffer-field1	**Description**
Logical Field Name (LU1)	Lookup Buffer Field

Comparison defines the type of query and can be one of the following:

Comparison	**Description**
EQ	Contents of *buffer-field1* is equal to the contents of *buffer-field2*
LT	Contents of *buffer-field1* is less than the contents of *buffer-field2*
LE	Contents of *buffer-field1* is less than or equal to the contents of *buffer-field2*
GT	Contents of *buffer-field1* is greater than the contents of *buffer-field2*
GE	Contents of *buffer-field1* is greater than or equal to the contents of *buffer-field2*

Buffer-field2 specifies the buffer holding the value to search on and can be one of the following:

Buffer-field2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
String	ASCII string delimited by double quotes Must be one character long
Number	Number prefixed by a number sign (#)

If the query is successful and finds the record, the pointer is set to that record.

Rule: *Buffer-field1* and *buffer-field2* must be the same type. For example, if *buffer-field1* is numeric, *buffer-field2* must be numeric.
If the record is not found, the pointer is undefined. The script must execute the command again to ensure a valid record pointer.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exception is as follows:

If the search fails to find the requested field or it detects end of file, control passes to *invalid label*.

Example This example searches the CONTROL_ID field for an exact match with the Input Buffer's contents. If there is no match, control passes to *ERROR_ID. Otherwise, control passes to the *PROCESS_ID.

```
QUERY CONTROL_ID,EQ,INPUT,*ERROR_ID,*PROCESS_ID
```

See Also READ
SEEK

READ

Purpose Copies the current record from the lookup file into the appropriate working buffer.

Syntax READ *record* [*(table-name)*] [*,* [*invalid label*] [*, valid label*]]

Process The READ command copies the current record into the appropriate working buffer, specified by *record*. After the script copies the current record into the buffer, the pointer advances to the next *record* in the file. *Table-name* selects which lookup table to read.

Record can be one of the following:

Record	**Description**
LOOKUP	Copies the Lookup Table record into the Lookup Buffer

Rule: A successful read increments the file pointer to the next record.

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exception is as follows:

If there is no record to read or the current record is pointing to a different record type, and *invalid label* is defined, control passes to that label.

Example This example shows how the software copies the current lookup table record into the lookup table buffer. Control passes to the next line.

```
READ LOOKUP
```

See Also BSEARCH
QUERY
SEEK

RESTORESCREEN

Purpose	Re-displays a previously-saved screen.
Syntax	RESTORESCREEN
Process	The RESTORESCREEN command restores the contents of a previously saved screen to the screen, overwriting the current screen's contents. The SAVESCREEN command saved the original screen and stored it in the internal screen buffer.

Example This example restores contents of the original screen (containing "1234567890") to the screen, overwriting the screen's current contents.

```
CLEAR DISPLAY
MOVE "1234567890", DISPLAY
SAVESCREEN
CLEAR DISPLAY
MOVE "0987654321", DISPLAY
RESTORESCREEN
```

See Also SAVESCREEN

RETURN

Purpose	Breaks out of a subroutine.
Syntax	RETURN
Process	The RETURN command breaks out of a subroutine. It transfers control back to the command following the CALL activating the subroutine.
NOTE:	Using END in a subroutine also implies a RETURN. Therefore, the RETURN command is not required as the last command of a subroutine.

Example This example breaks out of a subroutine.

```
COMPARE FSIZE,EQ,#12,,*GOODDATA
RETURN
```

See Also CALL

RIGHT

Purpose — Extracts the right-most characters from a string.

Syntax — RIGHT *buffer-field1* , *buffer-field2* , *length* [*,* [*invalid label*] [*, valid label*]]

Process — The RIGHT command extracts the right-most characters from *buffer-field1*, specified by *length*, and copies them into *buffer-field2*.

The *buffer-field* fields can be one of the following:

Buffer-field1, 2* and *length	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Buffer-field1 only: String	ASCII string delimited by double quotes. Must be one character long
Buffer-field1 and *length* only: Number	Number prefixed by a number sign (#)

Rule: *Length* must be numeric.

Optional Fields — *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example 1 — This example extracts the five right-most characters from SHIP_NO and copies them to SKU.

```
RIGHT SHIP_NO,SKU,#5
```

Example 2 — This example extracts the NUMCHARS right-most characters from SHIP_NO and copies them to SKU.

```
RIGHT SHIP_NO,SKU,NUMCHARS
```

See Also

LEFT
LSTRIP
MID
RSTRIP

RSTRIP

Purpose	Strips characters from a field, and copies the remaining characters to another field.
Syntax	RSTRIP *buffer-field1, buffer-field2, length* [*,* [*invalid label*] [*, valid label*]]

Process

The RSTRIP command strips the right-most characters from *buffer-field1* and copies the remaining characters to *buffer-field2*. *Length* is the number of characters to strip.

Buffer-field1, *buffer-field2*, and length can be one of the following:

Buffer-field1, *Buffer-field2*, and *Length*	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Field-buffer1 and *Field-buffer2* only: String	ASCII string delimited by double quotes.
Length only: Number	Number prefixed by a number sign (#)

Optional Fields

Invalid and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example

This example strips the five right-most characters from the SHIP_NO field and copies the remaining characters to the SKU field.

```
RSTRIP SHIP_NO, SKU, #5
```

See Also

LSTRIP
CSTRIP
TSTRIP

SAVESCREEN

Purpose	Saves the contents of the current screen.
Syntax	SAVESCREEN
Process	The SAVESCREEN command moves the contents of the current screen to the internal screen buffer. The RESTORESCREEN command re-displays the saved screen.
NOTE:	The internal screen buffer is cleared (and therefore the screen is lost) when the READY prompt appears, you calibrate the printer, or the application ends.

Example This example displays “1234567890” on the screen and saves it.

```
CLEAR DISPLAY
MOVE "1234567890", DISPLAY
SAVESCREEN
```

See Also RESTORESCREEN

SEEK

Purpose

Positions the record pointer within the lookup table.

Syntax

SEEK *modifier* , *file-type* [*(table-name)*] [*,* [*invalid label*] [*, valid label*]]

Process

The SEEK command positions the record pointer within the lookup table, according to *modifier*.

Modifier specifies the current record's placement and can be one of the following:

Modifier	**Description**
NEXT	Advance to next record
PREVIOUS	Move to previous record
START	Reset to beginning of file
END	Advance to last record

File-type specifies the type of file and can be one of the following:

File-type	**Description**
LOOKUPFILE	Lookup Table File

Table-name selects which lookup table to seek.

Optional Fields

Invalid and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exceptions are as follows:
If the NEXT modifier advances the current record past the end of the file, or the PREVIOUS modifier moves the current record before the beginning of the file, control passes to *invalid label* (if defined).|
When the selected file is empty, any modifier triggers an end of file condition. Then, control passes to *invalid label* (if defined).

Example 1

This example advances the current record in the lookup table by one record, and if an end of file condition occurs, control passes to *EOF_LABEL.

```
SEEK NEXT,LOOKUPFILE,*EOF_LABEL
```

See Also

QUERY
READ

SUB

Purpose — Subtracts the contents of one field from the contents of another.

Syntax — SUB *buffer-field1* , *buffer-field2* [*,* [*invalid label*] [*, valid label*]]

Process — The SUB command subtracts the contents of *buffer-field1* from the contents of *buffer-field2*, inserting the result into *buffer-field2*.

The *buffer-field* fields can be one the following:

Buffer-field1* and *2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
Buffer-field1 only: Number	Number prefixed by a number sign (#)

Rule: Only numeric fields are allowed.

Optional Fields — *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exception is as follows:

If *buffer-field2* becomes negative and *invalid label* is defined, control passes to that label.

Example — This example subtracts the contents of CONTROL_ID from TEMP1. Then, control passes to the next line.

```
SUB CONTROL_ID,TEMP1
```

See Also — ADD
DEC
INC
MULTIPLY
DIVIDE

SWITCH

Purpose

Directs script flow by branching to a set of commands based on the value of a variable.

Syntax

```
SWITCH  buffer-field1
    CASE  buffer-field
    .
    .
    .
    CASE  buffer-field
    .
    .
    .
    DEFAULT
    .
    .
    .
ENDSWITCH
```

Process

The SWITCH command directs script flow by branching to a set of commands based on the value of a variable. The command compares *buffer-field1* to the *buffer-field* listed with each case command. If the fields are equal, the script executes the commands following the CASE command. Execution stops when the script reaches the next CASE, DEFAULT, or ENDSWITCH.

If no *buffer-field* fields match *buffer-field1*, the script executes the set of commands after DEFAULT.

NOTE: There is no BREAK command to terminate CASE blocks, so this command does not support CASE fall-through. Also, **Do Not** use SWITCH inside a macro.

The *buffer-field* fields can be one of the following:

Buffer-field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Number	Number prefixed by a number sign (#)
String	ASCII string delimited by double quotes. Must be one character long

Example This example compares the Input Buffer's contents to <<, >>, and =. For example, if the input contains >>, the script executes the commands following CASE ">>" until the next CASE or DEFAULT command. Control then passes to ENDSWITCH. If the input buffer does not match any values, the script executes the commands following DEFAULT, until it reaches ENDSWITCH.

```
SWITCH INPUT
   CASE "<<"
         CALL SCROLLUP
         CALL DISPLAYMENU
   CASE ">>"
         CALL SCROLLDOWN
         CALL DISPLAYMENU
   CASE "="
         CALL SELECTMENUITEM
   DEFAULT
         BEEP
ENDSWITCH
```

See Also IF
WHILE

SYSSET

Purpose Sets the printer's default parameters.

Syntax SYSSET *function*, *parameter1*, *parameter2*

Process The SYSSET command sets the *function* parameter with the *parameter1* value, and if applicable, the *parameter2* value. If parameter1 and parameter2 are both buffer fields, they must be numeric. If they are a constant, precede it with a number sign (#) except where noted.

The *function, parameter1, and parameter2* fields can be one of the following:

Function/Description	***Parameter1***	***Parameter2***
PROMPTS The language to use for the printer's prompts.	**1** (English) **2**, **3** (Downloaded Foreign) **4** (Alternate)	n/a
BAUDRATE The rate for data transfers.	**19.2K**, **9600**, **4800**, **2400**, **1200** (Do Not precede with #)	n/a
FLOWCONTROL The flow control for data transfers.	**NONE**, **DTR**, **RTSCTS**, **XONOFF** (Do Not enclose in quotes)	n/a
PARITY The parity for data transfers.	**ODD**, **EVEN**, **MARK**, **SPACE**, **None** (Do Not enclose in quotes)	n/a
STATUSPOLLING Perform status polling during data transfers.	**0** (Disabled) **1** (Enabled)	n/a
STOPBIT The number of stop bits for data transfers.	**1** or **2**	n/a
DATABITS The number if data bits for data transfers.	**7** or **8**	n/a
ONDEMAND Print labels only when requested.	**0** (Disabled) **1** (Enabled)	n/a
BACKLIGHT Enables or disables this parameter. If enabled, sets the number of seconds without activity before the backlight turns off automatically.	**0** (Disabled) **1** (Enabled) **2-480** (Timeout)	n/a

Function/Description	***Parameter1***	***Parameter2***
SHUTDOWN Enables or disables this parameter. If enabled, sets the number of seconds without activity before the printer turns off automatically.	**0** (Disabled) **1** (Enabled) **2-480** (Timeout)	n/a
LABEL The label's dimensions in dots.	Width **208** (1.2") or use this formula: **192 * width in inches - 33**	Length 89 (.55") or use this formula: **192 * length in inches - 32**
REVVID Enables or disables reverse video on the screen.	**0** (Disabled) **1** (Enabled)	n/a
STATUSPOLLCCHAR Enables or disables status polling and specifies the character.	**0** (Disabled) **1** (Enabled)	The character to use. The default is 05H.
IMMEDCMD Enables or disables the processing of immediate commands and specifies the character.	**0** (Disabled) **1** (Enabled)	The character to use. The default is '^'.

Example This example specifies to use English prompts.

```
SYSSET PROMPTS, #1
```

TOKEN

Purpose	Sets a token delimiter or extracts a token-delimited sub-field from a larger field.

Syntax: To set a token delimiter:

TOKEN DELIMETER *character* [*,* [*invalid label*] [*, valid label*]]

To extract a sub-field:

TOKEN *buffer-field1*, *buffer-field2* [*,* [*invalid label*] [*, valid label*]]

Process: The TOKEN command sets *character* as the token delimiter or extracts a sub-field from *buffer-field1* (delimited by *character*) and places it in *buffer-field2*. Subsequent calls to this command using the same fields returns the next sub-string. You must set the delimiter before extracting sub-fields (the default is a comma).

Character, buffer-field1 and buffer-field2 can be one of the following:

Character, buffer-field1 and buffer-field2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field
SCRATCH	Scratch Buffer Field
String	An ASCII string delimited by double quotes
Number	A number prefixed by a number sign (#)

Optional Fields: *Invalid* and *valid labels* are discussed in "Script Flow" at the beginning of this chapter. The exception is as follows:

Example This example sets the token delimiter to *. Then, it extracts the strings PAXAR and CORPORATION from TASOURCE and moves them to the printer's screen one at a time.

```
   MOVE "PAXAR*CORPORATION", TASOURCE
   TOKEN DELIMITER, "*"
*GETTOKEN
   TOKEN TASOURCE, TATOKEN, *DONE
   MOVE TATOKEN, DISPLAY
   JUMP *GETTOKEN
*DONE
```

TSTRIP

Purpose	Strips characters from a field based on a template.
Syntax	TSTRIP *buffer-field1, buffer-field2* [*,* [*invalid label*] [*, valid label*]]

Process

The TSTRIP command strips data from *buffer-field1* as specified by *buffer-field2*.

Buffer-field2 contains a template that has a series of numbers and underscore characters (_). The printer matches the *buffer-field1* with the template, resulting in new data, as follows:

- ♦ If the characters in the same position match, they are stripped.
- ♦ If the template has an underscore character, the printer does not strip that character.
- ♦ If the character is the same position do not match, they are are not stripped.

Buffer-field1 and buffer-field2 can be one of the following:

Buffer-field1 and Buffer-field2	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Buffer-field2 only: String	An ASCII string delimited with double quotes.

Optional Fields

Invalid and *valid labels* are discussed in "Script Flow" at the beginning of this chapter.

Example

```
TSTRIP SHIP_NO, "1___66"
```

In this example, assume "123456" is in the SHIP_NO field. It matches up to the template as follows:

Original Data	123456
Template	1___66
New Data	2345

Position	**Match Description**
1	1 matches 1, so the number is stripped.
2	Underscore keeps the 2.
3	Underscore keeps the 3.
4	Underscore keeps the 4.
5	5 does not match 6, so the number is kept.
6	6 matches 6, so the number is stripped.

See Also

CSTRIP
RSTRIP
LSTRIP

UPPER

Purpose Converts the specified field to upper-case characters.

Syntax UPPER *source*

Process The UPPER command converts *source* to upper-case characters.

Source can be one of the following:

Source	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Logical Field Name (Field1) [Index]	Array Buffer Field

Example This example converts any lower-case characters in TEMP_SKU to upper-case.

```
UPPER TEMP_SKU
```

See Also LOWER

VALIDATE

Purpose	Validates a check digit based on check digit scheme.
Syntax	VALIDATE *source, type* [, [*invalid label*] [*, valid label*]]
Process	The VALIDATE command validates the check digit in *source*, based on the check digit scheme specified by *type*.

Source and *Type* can be one of the following:

Source and Type	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Number	Number prefixed by a number sign (#)
Source only: String	An ASCII string delimited with double quotes.
Logical Field Name (Field1) [Index]	Array Buffer Field

Type must be prefixed with a # sign and can have one of the following values:

ID	**Scheme**	**ID**	**Scheme**	**ID**	**Scheme**
#1	Reserved	#9	Custom Check Digit 5	#17	EAN8 Check Digit
#2	Sum of Digits	#10	Custom Check Digit 6	#18	EAN13 Check Digit
#3	Sum of Products	#11	Custom Check Digit 7	#19	LAC Check Digit
#4	Reserved	#12	Custom Check Digit 8	#20	Code 39 Check Digit
#5	Custom Check Digit 1	#13	Custom Check Digit 9	#21	MSI Check Digit
#6	Custom Check Digit 2	#14	Custom Check Digit 10	#22	Postnet
#7	Custom Check Digit 3	#15	UPCA Check Digit	#23	UPC+Price CD
#8	Custom Check Digit 4	#16	UPCE Check Digit	#24	EAN+Price CD

Optional Fields *Invalid* and *valid labels* are discussed in "Script Flow" earlier in this chapter.

Example In this example, the printer validates the check digit in the Input Buffer by using the Sum of Digits check digit scheme.

```
VALIDATE INPUT #2
```

See Also LOWER

WHILE

Purpose

Repeats a sequence of commands as long as a condition is true.

Syntax

```
WHILE  buffer-field1 comparison buffer-field2
.
.
.
ENDWHILE
```

Process

The WHILE command repeats a sequence of commands as long as a condition is true.

If the condition is true, the script executes the commands listed between WHILE and ENDWHILE. When script reaches ENDWHILE, it checks the condition again. If the condition still exists, it executes the commands again. If the condition is false, the script branches to the line after ENDWHILE.

Buffer-field1 and *buffer-field2* are the compared items in the condition. They can be one of the following:

Buffer-field	**Description**
INPUT	Input Buffer
Logical Field Name (TEMP1)	Temporary Buffer Field
Logical Field Name (LU1)	Lookup Buffer Field
Logical Field Name (PR1)	Printer Buffer Field
Number	Number prefixed by a number sign (#)
String	ASCII string delimited by double quotes. Must be one character long

Comparison is the operator used to compare *buffer-field1* and *buffer-field2*. It can be one of the following:

Comparison	**Description**
=	Equals
==	Equals
<>	Not equal
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

NOTE: Use the BREAK command to break out of a WHILE loop prematurely. For example, you could use it when an error occurs. Also, **Do Not** use WHILE inside a macro.

Example This example calls the macro PTRIDLE that checks for the printer status. The WHILE loops executes until the printer is ready to accept more data.

```
MOVE #0, tPrinterOK
  WHILE tPrinterOK == #0
    PTRIDLE   tPrinterOK
  ENDWHILE
```

See Also IF
SWITCH

SAMPLE SCRIPT

A

This chapter provides a sample script for retail printing. Depending on the character entered by the user, a different format prints. One is a compliance format, another is a receiving format, and the other is a sale format. Use this script as a guide when you program the printer. See Chapter 4, “Program Structure” for additional programming tips.

```
;Script File
;Sample Script
;Author:  A.Kramer
;Date:    Sept. 6, 2001

;This sample prints one of three formats, depending on the character
;entered by the user.

Define SCRATCH, 5000, A
AUTOSTART

Function Start
Begin
APPVERSION "AnyStore","V1.0"
call SendFmt
call main
End

Function Main
Begin
*Moredata
fetch comm
switch input
     case "C"
       call Comply

     case "R"
       call Receiving

     case "S"
       call Sale

     default
       clear input
     endswitch

jump *Moredata

End
```

```
;The Comply function contains the batch data for the compliance format.

Function Comply
Begin
     MOVE "{B,1,N,1 |8,~03466598~034|", SCRATCH
     CONCAT "9,~0340~034|", SCRATCH
     CONCAT "10,~03436~034|", SCRATCH
     CONCAT "11,~0342508-09505~034|", SCRATCH
     CONCAT "12,~034950330~034|", SCRATCH
     CONCAT "13,~034FISHING ROD~034|", SCRATCH
     CONCAT "14,~034OH 45001~034|", SCRATCH
     CONCAT "16,~034LIMA~034|", SCRATCH
     CONCAT "17,~034123 US 35~034|", SCRATCH
     CONCAT "18,~034MYSTORE~034|", SCRATCH
     CONCAT "29,~0348~034|", SCRATCH
     CONCAT "30,~0340000028028665988~034|}", SCRATCH
     parse
     clear INPUT
     return
End

;The Receiving function contains the batch data for the receiving format.

Function Receiving
Begin
     move "{B,2,N,1|1,~034674148022201~034|", SCRATCH
     CONCAT "2,~034BULK TOMATO PASTE~034|}", SCRATCH
     parse
     clear INPUT
     return
End

;The Sale function contains the batch data for the sale format.

Function Sale
Begin
     move "{B,3,N,1|1,~0340632253993005~034|", SCRATCH
     CONCAT "2,~034SWEATER~034|", SCRATCH
     CONCAT "3,~034SMALL~034|}", SCRATCH
     parse
     clear INPUT
     return
End
```

```
;The SendFmt function moves the three formats into the scratch buffer.
;The batch data is sent when the user sends a "C," "R," or "S" character.

Function SendFmt
Begin
     move "{F,1,A,R,G,1218,0812,~034Comply~034|", SCRATCH
     CONCAT "L,S,89,59,89,749,16,~034~034|", SCRATCH
     CONCAT "L,S,341,59,341,749,16,~034~034|", SCRATCH
     CONCAT "L,S,440,13,440,796,6,~034~034|", SCRATCH
     CONCAT "L,S,947,13,947,796,7,~034~034|", SCRATCH
     CONCAT "L,S,1205,356,950,356,6,~034~034|", SCRATCH
     CONCAT "L,S,643,13,643,796,6,~034~034|", SCRATCH
     CONCAT "T,7,6,V,45,257,0,3,1,1,B,L,0,0|", SCRATCH
     CONCAT "R,1,~034028028~034|", SCRATCH
     CONCAT "T,8,5,V,45,468,0,3,1,1,B,L,0,0|", SCRATCH
     CONCAT "T,9,1,V,45,124,0,3,1,1,B,L,0,0 |", SCRATCH
     CONCAT "T,10,8,V,592,325,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "T,11,10,V,700,417,0,50,20,20,B,L,0,0 |", SCRATCH
     CONCAT "T,12,6,V,781,346,0,50,20,20,B,L,0,0|", SCRATCH
     CONCAT "T,13,40,V,500,51,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "T,14,20,V,971,376,0,50,14,12,B,L,0,0 |", SCRATCH
     CONCAT "T,15,19,V,998,11,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034MIAMISBURG OH 45342~034|", SCRATCH
     CONCAT "T,16,20,V,1022,376,0,50,14,12,B,L,0,0 |", SCRATCH
     CONCAT "T,17,20,V,1073,376,0,50,14,12,B,L,0,0 |", SCRATCH
     CONCAT "T,18,20,V,1124,376,0,50,14,12,B,L,0,0 |", SCRATCH
     CONCAT "T,19,16,V,1038,11,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034170 MONARCH LANE~034|", SCRATCH
     CONCAT "T,20,18,V,1079,11,0,50,12,10,B,L,0,0|", SCRATCH
     CONCAT "R,1,~034WORLD HEADQUARTERS~034|", SCRATCH
     CONCAT "T,21,17,V,1120,11,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034PAXAR CORPORATION~034|", SCRATCH
     CONCAT "T,22,13,V,592,51,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034SELLING UNIT:~034|", SCRATCH
     CONCAT "T,23,13,V,700,21,0,50,15,15,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034VENDOR/STYLE:~034|", SCRATCH
     CONCAT "T,24,8,V,876,41,0,50,24,20,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034MYSTORE~034|", SCRATCH
     CONCAT "T,25,10,V,782,21,0,50,15,15,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034PO NUMBER:~034|", SCRATCH
     CONCAT "T,26,5,V,1180,11,0,50,12,10,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034FROM:~034|", SCRATCH
     CONCAT "T,27,3,V,1174,376,0,50,14,12,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~034TO:~034|", SCRATCH
     CONCAT "T,28,1,V,45,191,0,3,1,1,B,L,0,0 |", SCRATCH
     CONCAT "R,1,~0340~034|", SCRATCH
     CONCAT "T,29,1,V,45,650,0,3,1,1,B,L,0,0 |", SCRATCH
     CONCAT "B,30,16,V,110,102,3,5,226,8,L,0 |", SCRATCH
     CONCAT "R,50,4,12 |}", SCRATCH
     CONCAT "{F,2,A,R,E,200,400,~034Receive~034|", SCRATCH
     CONCAT "B,1,12,F,92,110,4,12,50,8,L,0|", SCRATCH
     CONCAT "C,165,27,0,50,9,9,A,L,0,0,~034LOT# 6741~034,1|", SCRATCH
     CONCAT "C,166,238,0,50,9,9,A,L,0,0,~034QTY 48~034|", SCRATCH
     CONCAT "C,75,107,0,510,1,1,B,L,0,0,~034744148022201~034|", SCRATCH
     CONCAT "C,52,132,0,50,8,8,A,L,0,0,~03402/22/01  15:29~034,1|", SCRATCH
     CONCAT "T,2,20,V,29,123,0,50,8,8,A,L,0,0,1|}", SCRATCH
     CONCAT "{F,3,A,R,E,300,200,~034Sale~034|", SCRATCH
     CONCAT "B,1,13,F,99,52,7,2,40,7,L,0|", SCRATCH
     CONCAT "C,279,28,0,510,1,1,B,L,0,0,~034063   DEPT#25~034|", SCRATCH
     CONcaT "T,2,15,V,243,61,0,50,10,10,A,L,0,0,1|", SCRATCH
     CONCAT "T,3,8,V,215,71,0,50,10,10,A,L,0,0,1|", SCRATCH
     CONCAT "C,187,78,0,50,10,10,A,L,0,0,~034RED~034,1|", SCRATCH
     CONCAT "C,162,51,0,50,10,10,A,L,0,0,~034COTTON-RAMIE~034,1|", SCRATCH
     CONCAT "C,75,14,0,50,11,11,A,L,0,0,~034WAS     $39.99~034,1|", SCRATCH
     CONCAT "C,46,16,0,50,11,11,A,L,0,0,~034NOW      $30.00~034,1|}", SCRATCH
     parse
     return
End
```

INDEX

Visit **www.paxar.com** for sales, service, supplies, information, and telephone numbers for our locations throughout the world.

TOLL FREE:
1-800-543-6650 (In the U.S.A.)
1-800-363-7525 (In Canada)